VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV POČÍTAČOVÉ GRAFIKY A MULTIMÉDIÍ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF COMPUTER GRAPHICS AND MULTIMEDIA

SLUŽBA PRO STREAMOVÁNÍ HUDBY

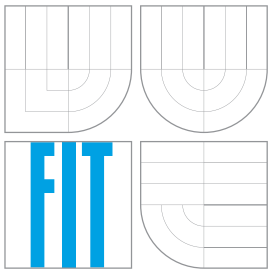
BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE
AUTHOR

ERIK ŠKULTÉTY

BRNO 2013

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY



FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV POČÍTAČOVÉ GRAFIKY A MULTIMÉDIÍ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF COMPUTER GRAPHICS AND MULTIMEDIA

SLUŽBA PRO STREAMOVÁNÍ HUDBY

MUSIC STREAMING SERVICE

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE
AUTHOR

ERIK ŠKULTÉTY

VEDOUCÍ PRÁCE
SUPERVISOR

Ing. JOZEF MLÍCH

BRNO 2013

Abstrakt

Tato práce se zabývá problematikou streamované hudby a následně pak implementací vzorové aplikace pro mobilní zařízení běžící na platformě MeeGo. Tato aplikace se v základu opírá o služby protokolu RTP, který je dostupný ve formě pluginů knihovny GStreamer. V rámci této práce je dále diskutována problematika cloud computingu a posléze implementována cloudová služba typu SaaS, která poskytuje klientské části aplikace seznam aktuálně dostupných rádií, přičemž tato služba rovněž uživatelům poskytuje přidávání nových rádií do internetové databáze.

Abstract

This thesis deals with music streaming subject and implementation of a model application for MeeGo based platforms. Application relies on services provided by RTP protocol. These services are supplied by GStreamer framework plugins. Cloud computing topic is also discussed later in this thesis, focussing mainly on the SaaS method. Finally, a server part which provides a list containing available radios, needed by the client part, as well as the possibility for users to add new radios into the database, is described.

Klíčová slova

Cloud computing, Software as a Service, OpenShift, audio streaming, rádio, hudba, MeeGo

Keywords

Cloud computing, Software as a Service, OpenShift, audio streaming, radio, music, MeeGo

Citace

Erik Škultéty: Služba pro streamování hudby, bakalářská práce, Brno, FIT VUT v Brně, 2013

Služba pro streamování hudby

Prohlášení

Prohlašuji, že jsem tuto bakalářskou práci vypracoval samostatně pod vedením pana Ing. Jozefa Mlícha.
Další informace mi poskytl Ing. Jaroslav Řezník.
Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

.....
Erik Škultéty
15. května 2013

Poděkování

Rád by som poďakoval vedúcemu práce Ing. Jozefovi Mlíchovi za vedenie práce a poskytnutie užitočných rád. Ďalej by som chcel poďakovať Ing. Jaroslavovi Řezníkovi za technické rady, návrhy, a taktiež za poskytnutie mobilného telefónu na testovanie aplikácie.

Obsah

1 Úvod **2**

2 Real-time Transport Protocol **3**
2.1 Úvod do RTP . . . 3
2.2 Princípy RTP protokolu . . . 4
2.3 RTP špecifikácia . . . 4
2.4 Zaistenie kvality služieb QoS . . . 6

3 Cloud computing **9**
3.1 Úvod do cloud computingu . . . 9
3.2 Cloud komponenty . . . 10
3.3 Služby . . . 12
3.4 Virtualizácia . . . 17

4 Návrh a implementácia aplikácie **20**
4.1 Návrh a implementácia užívateľského rozhrania . . . 20
4.2 Serverová časť - aplikácia typu SaaS . . . 26
4.3 Návrh aplikačného rozhrania, integrácia prehrávania do GUI . . . 30
4.4 Zhodnotenie výslednej aplikácie . . . 32

5 Záver **34**

A Konfiguračný súbor deploy **37**

B Konfiguračný súbor settings.py **38**

Kapitola 1

Úvod

Hudba bola vždy dôležitou súčasťou našej spoločnosti, dnes možno povedať, že dokonca neodmysliteľnou súčasťou nášho každodenného života, nakoľko sme s ňou neustále v kontakte, či už prostredníctvom rádia a s ním súvisiacimi stanicami alebo prostredníctvom prenosných mp3 prehrávačov.
Pojem rádia dnes chápeme skôr ako samotnú stanicu, avšak tento pojem vznikol iba odvodením od jeho fundamentálneho princípu operability, ktorým sú rádiové vlny. V tejto práci sa zameriavam na technológiu prehrávania vysielaní jednotlivých staníc s využitím internetového pripojenia a agregačného prostriedku v podobe serveru, ktorý uchováva zoznam dostupných staníc k prehrávaniu.

Vzhľadom k neustále rastúcej popularite internetového rádia je cieľom práce navrhnúť a vytvoriť mobilnú aplikáciu, ktorá by dokázala prehrávať jednotlivé stanice výhradne s využitím internetového pripojenia. Taktiež sa snaží predstaviť čitateľovi aktuálne používané technológie a metódy v kontexte multimediálneho prenosu cez internet, ako aj dokumentovať návrh a implementáciu samotnej aplikácie, realizujúcej prehrávanie rôznych dostupných rádiových staníc, bez nutnosti tieto stanice na internete vyhľadávať v podobe webových stránok. Primárnou platformou pre beh aplikácie boli zvolené mobilné zariadenia firmy Nokia s bežiacim operačným systémom MeeGo.

Text je logicky rozdelený do dvoch súvisiacich celkov, prvý celok je teoretický, ktorý pojednáva o zaužívaných technológiách a princípoch relevantných k zadaniu práce, poskytuje úvod do týchto technológií a ich stručný prehľad. Druhý celok slúži ako dokumentácia k praktickej časti tejto práce, ktorou je návrh aplikácie realizujúcej streamovanie hudobného obsahu. Kapitola 2 sa zaoberá základnými princípmi a funkcionalitou RTP protokolu, ktorý sa javí ako najvhodnejšia voľba pre prenos multimediálneho obsahu v reálnom čase po sieti. Ďalej sa v kapitole 3 venujem základom cloud computingu, s primárnym zameraním na typ SaaS. Následne v kapitole 4 je popísaný návrh samotnej aplikácie (klientskej i serverovej časti), použité technológie a nástroje, ako aj podrobný popis implementácie tejto aplikácie.

Kapitola 2

Real-time Transport Protocol

Táto kapitola si dáva za úlohu v prvej časti poskytnúť krátky úvod do protokolu RTP, jeho vzniku a použitiu, v ďalšej sekcii 2.2 potom poukázať na fundamentálne princípy v návrhu RTP protokolu (application level framing a end-to-end-princíp). V tretej časti 2.3 sa snaží poskytnúť čitateľovi stručný popis špecifikácie RTP protokolu, predovšetkým však zdôrazňuje formát RTP paketu, a taktiež vysvetľuje dôležitý pojem *payload formáty*. Záver kapitoly 2.4 sa zaoberá zaistením kvality služieb a popisu jej dvoch najrozšírenejších modelov - integrovaných a difrenciovaných služieb. Táto kapitola čerpá poznatky zo zdrojov [11, 14, 16, 7].

2.1 Úvod do RTP

Kľúčovým štandardom pre prenos multimediálneho obsahu v IP sieťach je Real-time Transport Protocol, sprevádzaný spolu so špecifikovanými profilmi a payload formátmi. Medzi služby, ktoré tento protokol zahŕňa, patrí timing recovery, detekcia straty a opravy paketov, payload a source identifikácia, reception quality feedback, media synchronization, membership management. RTP bolo pôvodne navrhnuté pre použitie v multicastových konferenciách, použitím odľahčeného relačného modelu. Odvtedy sa RTP preukázal byť ako vhodným kandidátom pre široké použitie v rôznych multimediálnych aplikáciách: H.323 video konferencie, wecasting, distribúcia TV signálu. Protokol je vhodný pre použitie od P2P sietí až po multicast sedenia tisíciek užívateľov, v telefónnych sieťach s malým garantovaným prenosovým pásmom až po prenos nekomprimovaného HDTV signálu v gigabitových rýchlostiach.

Jedným z najdôležitejších požiadavkov na transportný protokol je spoľahlivé doručenie všetkých paketov po sieti. Samozrejme, vždy je snaha o spoľahlivé doručenie bez akýchkoľvek strát, avšak mnohé audio či video aplikácie dokážu tolerovať určitú strátovosť, pretože efekt samotnej strátovosti je takmer ihneď eliminovaný príchodom nových dát multimediálneho toku. Miera tolerancie strátovosti je variabilná a závisí primárne na konkrétnej aplikácii, použitej kódovacej metóde a na modele strátovosti. Tieto vlastnosti sa týkajú kvality zaistenia služieb, ktorá je popísaná v 2.4.

Hlavnou výhodou použitia IP ako sprostredkovateľa prenosu real-time audia a videa je skutočnosť, že poskytuje unifikovanú a konvergentnú sieť, to znamená, že tú istú sieť možno využiť pre prenos hlasu, hudby, videa a zároveň využívať služby ako e-mail, webové služby, FTP služby, hry a iné. Výsledkom je značná úspora nákladov z pohľadu infraštruktúry, nasadenia, podpory a správy.

2.2 Princípy RTP protokolu

RTP protokol poskytuje mechanizmus, ktorý je schopný doručiť obrovský objem real-time mediálnych dát postavený nad nespoľahlivou transportnou vrstvou referenčného modelu ISO/OSI. Tento mechanizmus sa podarilo dosiahnuť návrhom vychádzajúcim z príbuzných filozofií *application-level framing* a *end-to-end principle*. Celá nasledujúca sekcia sa opiera najmä o poznatky z [11].

Fundamentálnou myšlienkou filozofie *application-level framing* je skutočnosť, že iba samotná aplikácia má dostatočnú znalosť o svojich dátach, aby bola schopná rozhodnúť, akým spôsobom budú dáta transportované. Dôsledkom tohto je, že transportný protokol by mal akceptovať dáta reprezentované v aplikačných dátových jednotkách (ADU) a odkrývať všetky dostupné podrobnosti o ich doručení. Aplikácia na základe získaných údajov zvolí vhodnú reakciu v prípade vzniknutej chyby. Taktiež aktívne spolupracuje na transporte aby sa tak docielilo požadovaného spoľahlivého doručenia dát.
Filozofia pracuje so skutočnosťou, že existuje viac spôsobov, ako sa môže aplikácia vysporiadať s možnými problémami na sieti, a teda voľba prístupu a riešenia závisí jednak na samotnej aplikácii, ale aj na konkrétnej situácii, v ktorej sa má zvolený prístup uplatniť. V niektorých prípadoch je nevyhnutné preposlať identickú kópiu stratených dát, v iných prípadoch môže prísť k nežiadúcej zmene dát a je nutné preposlať náhradu za poškodené dáta, pričom náhrada sa bude od pôvodného originálu pochopiteľne líšiť, teda je nutné sa aj s týmto faktom vysporiadať. V určitých špecifických prípadoch je dokonca možné stratu ignorovať. Uvedené prístupy či riešenia sú možné iba ak aplikácia úzko spolupracuje s transportnou vrstvou.

Hlavná myšlienka application-level framing je v rozpore s návrhom protokolu TCP, ktorý skrýva nespoľahlivú povahu nižšej IP vrstvy, aby tak dosiahol spoľahlivého doručenia dát, aj za cenu oneskorenia dát. Naopak, UDP protokol je vhodná voľba ako transportný protokol vzhľadom k vlastnostiam real-time médií. UDP protokol nám umožňuje tolerovať straty, kde je to nevyhnutné/možné, ale zároveň nám dáva dostatočnú flexibilitu použiť celé spektrum zotavovacích techník, z ktorých časť bola spomenutá v predošlom odstavci. Z uvedeného vyplýva, že spomínané techniky dávajú aplikácii výhodu reakcie na vzniknutý problém na sieti vhodným spôsobom, v porovnaní s obmedzením, ktoré by bolo striktne dané transportnou vrstvou.

Ďalšou filozofiou, ktorá bola štandardom RTP adoptovaná je *end-to-end princíp*. Hlavným dôsledkom end-to-end princípu je skutočnosť, že inteligencia postupuje smerom nahor na sieťovom vrstvovom modele. Ak systémy tvoriace sieťovú cestu nikdy nebudú brať zodpovednosť za doručené dáta, nemusia byť zbytočne zložité a robustné. V prípade, že nie sú schopné dáta doručiť, dáta sú automaticky zahodené, pretože koncové systémy/aplikácie sa s tým musia dokázať vysporiadať. Všetka inteligencia a rozhodovacia logika je teda na koncových systémoch, nie v samotnej sieti ako takej. Pre porovnanie návrhu, v klasickej telefónnej sieti sa dáva prednosť modelu, kde je inteligencia odbúraná z koncových zariadení a naopak umiestnená na nižšie prenosové vrstvy a zariadenia na sieti.

2.3 RTP špecifikácia

RTP je protokolom relačnej vrstvy úzko spolupracujúci s transportným protokolom UDP, pričom efektívne dopĺňa jeho služby o mechanizmy detekcie straty dát, oznámenia

o kvalite doručenia dát adresátovi, rôzne časové a synchronizačné mechanizmy, identifikáciu a mapovanie médií na príslušné payload formáty, a iné. Z implementačného hľadiska je podľa Perkinsa v [11] protokol RTP často integrovaný do samotnej aplikácie, prípadne je jeho implementácia dostupná vo forme knižnice. Špecifikácia popisuje dva úzko spolupracujúce podprotokoly: data transfer protocol a riadiaci control protocol.
Data transfer protocol je zodpovedný za prenos real-time dát, pričom informácie poskytované týmto protokolom zahŕňajú časové razítka (timestamps) nutné pre synchronizáciu prenášaných dát, sekvenčné číslo (sequence number), dôležité pri detekcii straty paketov alebo ich preusporiadaní a payload formát dát, ktorý indikuje konkrétny formát, do ktorého boli dáta zakódované. Narozdiel od dátového protokolu, kontrolný protokol špecifikuje používanie spätnej väzby pre dodržiavanie kvality služieb QoS a taktiež pre zachovanie synchronizácie medzi jednotlivými dátovými prúdmi.
Pôvodná špecifikácia RTP vyžadovala pri sieťovej komunikácii dve po sebe idúce čísla portov, pričom párne čísla portov boli určené pre dátový protokol RTP a nasledujúce nepárne číslo pre kontrolný protokol. Posledná revízia štandardu však upustila od týchto požiadavkov a už nie je nevyhnutné, aby boli porty využívané dátovým protokolom RTP číslované výhradne párnymi číslami. Na základe uvedeného je teraz možné využívať porty, ktorých číselné identifikácie sú od seba vzdialené, avšak vzhľadom k spätnej kompatibilite starších implementácii sa naďalej odporúča dodržiavať staršiu normu.

RTP payload formáty

Dôležitú súčasť RTP protokolu tvoria tzv. payload formáty, ktoré definujú ako budú dané typy mediálnych dát prenášané pomocou RTP. Samotné formáty sú odkazované a spravované RTP profilmi, pričom profilom sa myslí istý menný priestor, ktorý má za úlohu naviazať identifikátor typu mediálneho obsahu (payload type) obsiahnutom v RTP pakete na presnú špecifikáciu payload formátov. Dôsledkom tohto je, že aplikácia je schopná pred samotným prenosom mediálneho obsahu tieto dáta asociovať k príslušnému mediálnemu kodeku. Payload formát špecifikuje použitie príslušných polí v hlavičke RTP paketu (obrázok 2.1), v niektorých prípadoch tiež definuje doplňujúcu payload hlavičku, ktorá sa v RTP pakete umiestňuje za hlavnú RTP hlavičku paketu. Následne je výstup vygenerovaný mediálnym kodekom zapuzdrený hneď do niekoľkých RTP paketov, pričom niektoré časti výstupu sú mapované na polia RTP hlavičky paketu alebo sú vložené do doplnkovej payload hlavičky a najväčšia časť sa mapuje do dátovej časti paketu. Dôvodom existencie ďalšej doplňujúcej payload hlavičky je rôznorodosť implementácií kodekov na strane prijímateľa, ktoré pre správnu činnosť vyžadujú dodatkové informácie, obsiahnuté práve v doplnkovej payload hlavičke.

bit offset	0-1	2	3	4-7	8	9-15	16-31
0	Version	P	X	CC	M	PT	Sequence Number
32	Timestamp						
64	SSRC identifier						
96	CSRC identifiers ...						
96+32*CC	Profile-specific extension header ID						Extension header length
128+32*CC	Extension header (optional) ...						
Payload data							

Obrázek 2.1: RTP paket (podľa [11])

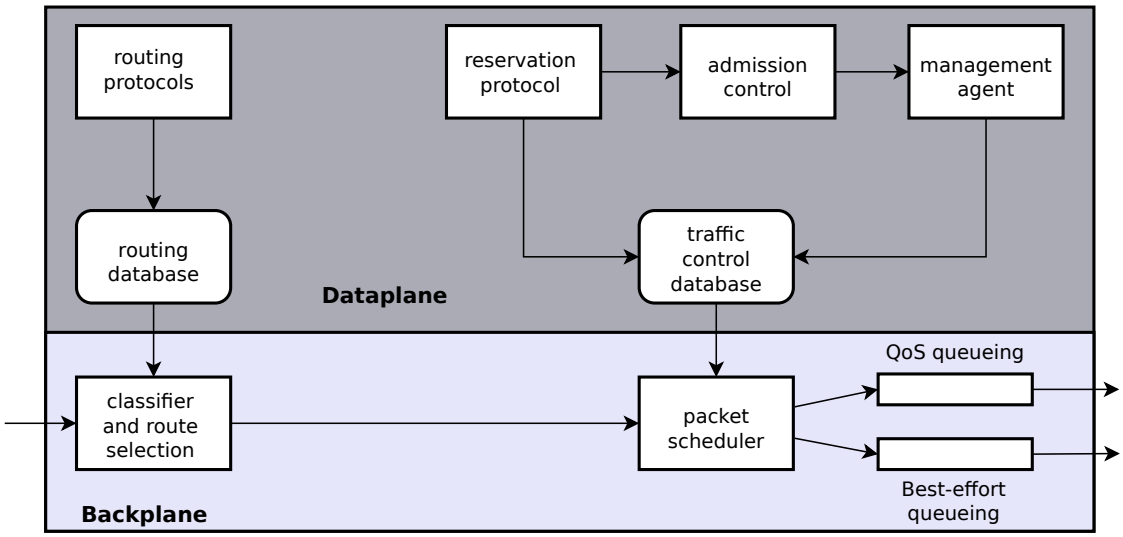
2.4 Zaistenie kvality služieb QoS

Napriek tomu, že RTP bolo navrhnuté tak, aby sa spoľahlivo opieral o službu najlepšieho možného doručenia poskytovanú IP protokolom, je takmer nevyhnutné poskytnúť dátovému toku RTP zvýšenú kvalitu služieb, napr. možnosť rezervácie zdrojov na sieti pred samotným započatím dátového prenosu. V nasledujúcich častiach budú predstavené dva najrozšírenejšie spôsoby zaistenia kvality služieb, architektúra integrovaných služieb (označuje sa aj ako IntServ) a architektúra diferenciovaných služieb (DiffServ). Sekcia o integrovaných službách čerpá primárne z [2] a sekcia o diferenciovaných službách z [10]. Ďalšie doplnkové informácie poskytla kniha [7].

Integrované služby

Úlohou integrovaných služieb je zaistenie podpory QoS v sieťach IP, predovšetkým však riešenie problému zdieľania dostupnej kapacity sieťových zdrojov v dobe zahltenia. Integrované služby tvorí niekoľko kľúčových častí, konkrétne sa jedná o rezerváciu zdrojov, riadenie prístupu, spôsob riadenia front a spôsob zahadzovania paketov. Príklad implementácie integrovaných služieb je na obrázku 2.2. Hardvérová časť smerovača backplane, ktorou prechádzajú všetky pakety, je optimalizovaná na rýchlosť a prebieha v nej klasifikácia paketov a výber cesty k smerovaniu. Plánovač paketov (Packet Scheduler) obsluhuje jednu či viacero front pre každý výstupný port. Určuje poradie, v akom budú pakety z front odoberané, prípadne či je ich nutné zahodiť. Rezervácia výpočetných zdrojov, správa zdrojov a prideľovanie prebieha v časti dataplane.
Na základe klasifikácie je určená trieda prevádzky pre daný IP datagram, pričom táto klasifikácia je uskutočnená na základe dát z hlavičky IP datagramu. Konkrétna trieda môže odpovedať jednému či viacerým tokom. Rezervácia výpočetných prostriedkov u architektúry integrovaných služieb prebieha na základe rezervačného protokolu RSVP, ktorý je zodpovedný za alokáciu prostriedkov pre jednotlivé dátové toky v sieti (per-flow reservation). Toto riešenie na jednu stranu umožňuje garantovať kvalitu služieb pre jednotlivé toky, avšak zároveň tento model so sebou prináša aj určité problémy, z ktorých najvýznamnejším je rozšíriteľnosť. Pretože sa jedná o rezerváciu per flow, každý tok prechádzajúci smerova-

čom vyžaduje spracovanie rezervačného požiadavku a alokáciu stavu toku. Predovšetkým pre chrbtové smerovače je takáto alokácia zdrojov a výpočetná náročnosť neúnosná. Ďalším problémom architektúry integrovaných služieb je pôvodne plánované použitie pre malý počet predom špecifikovaných tried služieb. Táto množina tried kvality neumožňuje popísať kvalitatívne či vzťahové požiadavky, kedy služba jednej triedy bude mať prednosť pred službou inej triedy. Na tieto požiadavky však reaguje model diferenciovaných služieb popísaný v nasledujúcej sekcii.



Obrázek 2.2: Implementácia integrovaných služieb v smerovači [2]

Diferenciované služby

Ako bolo spomenuté v predošlej sekcii, hlavným problémom integrovaných služieb je rozšíriteľnosť. Naproti tomu architektúra DiffServ sa zameriava práve na rozšíriteľnosť modelu a flexibilitu, konkrétne schopnosť pracovať s rôznymi triedami prevádzky. Rozšíriteľnosť počíta s použitím tejto architektúry aj na chrbtových sieťach, kde cez smerovače prechádzajú stovky tisícov tokov. Z tohto dôvodu sa architektúra zameriava na hraničné smerovače, kde sa vykonávajú výpočetne náročnejšie operácie, zatiaľ čo chrbtové smerovače vykonávajú skôr jednoduchšie funkcie. Architektúru diferenciovaných služieb tvoria dva typy prvkov: hraničné prvky (klasifikácia, podmienky prevádzky) a chrbtové prvky (preposielanie).

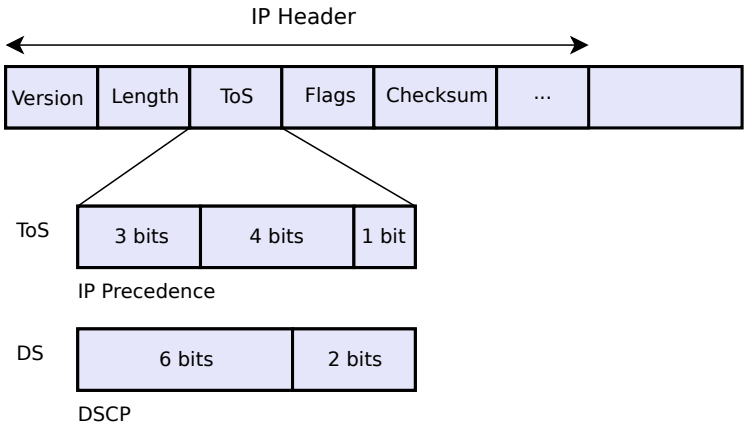
Hraničnými prvkami sú buď koncové zariadenia alebo najbližšie smerovače, ktoré podporujú diferenciované služby. Tieto hraničné prvky značkujú prechádzajúce pakety, konkrétne nastavením hodnoty DS (Differentiated Services) v hlavičke IP datagramu. Značenie, ktoré daný paket dostane, identifikuje triedu prevádzky, do ktorej patrí. Rôzne triedy budú mať rôzne značenia. Po označení paketu môže byť tento buď poslaný do siete, vložený do radu a čakať na odoslanie alebo môže byť zahodený.

Keď príde označený paket na chrbtový smerovač s podporou diferenciovaných služieb, je preposlaný na ďalší uzol podľa definovaného správania prislúchajúceho jeho triede (Per-hop Behavior, tiež PHB). Nastavené správanie PHB ovplyvňuje zdieľanie pamäte na smerovači a prenosového pásma rôznymi triedami prevádzky. Podstatné na architektúre DiffServ je,

že na chrbtovom smerovači sa spracovanie paketu riadi výhradne jeho značením, t.j. triedou prevádzky, do ktorej paket patrí.

Klasifikácia prevádzky pomocou DSCP

U diferenciovaných služieb je značka paketu uložená v poli DS (Differentiated Services) v hlavičke protokolu IP. Pôvodná definícia protokolu IP však toto pole neobsahuje, ale obsahuje osembitové pole Typ of Service. Neskoršie štandardy zaviedli inú interpretáciu tohto poľa. Nazývajú ho DS. Toto pole sa skladá zo šesťbitového kódu DSCP (Differentiated Service Code Point), ktorý definuje typ správania PHB (Per-hop Behavior), a dvojbitovej hodnoty CU (Currently Unused), ktorá sa momentálne nepoužíva (obrázok 2.3). Nastavenie hodnoty DSCP vykonáva hraničný prvok siete na základe klasifikácie prevádzky. Táto klasifikácia môže byť uskutočnená na základe rôznych kritérií najčastejšie však podľa zdrojovej, respektíve cieľovej adresy, čísla portu, typu protokolu a iných.



Obrázek 2.3: IP precedencia a DS pole v hlavičke IP datagramu [10, 7]

Táto kapitola položila teoretický základ do protokolu RTP. Tento základ nám poslúži v kapitole 4, kde za použitia knižnice *QtMultimediaKit* a pluginov sady knižníc (angl. *framework*) GStreamer bude tento protokol jadrom streamovaného prehrávania.

Kapitola 3

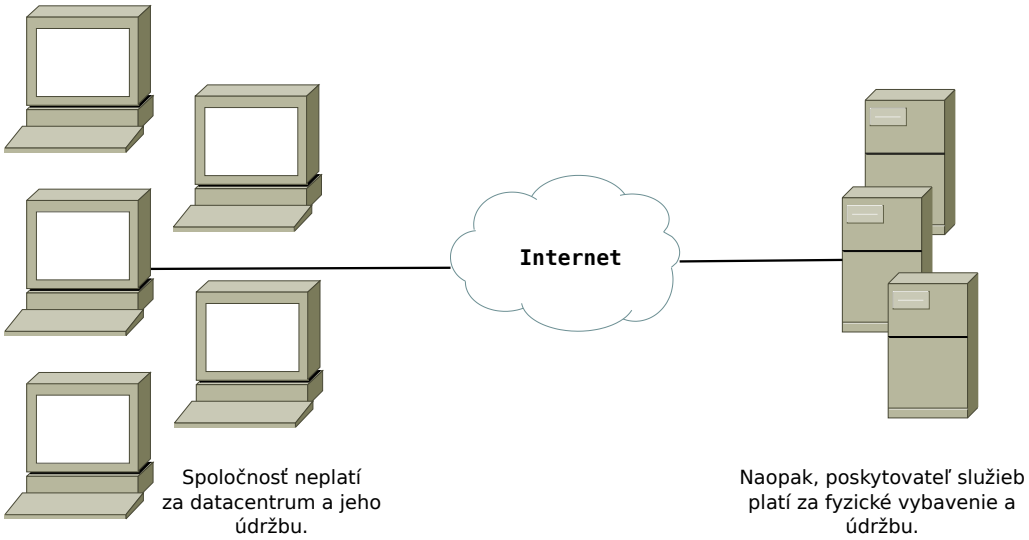
Cloud computing

Táto kapitola čitateľa v prvej časti stručne oboznámi so základmi cloud computingu, s dôležitými pojmami a princípmi jeho fungovania. V druhej časti sú zmenené fundamentálne komponenty [17] a následne sú v sekcii 3.3 popísané jednotlivé služby cloud computingu, predovšetkým však so zameraním na SaaS [17]. Na záver kapitoly je istý úsek venovaný pojmu virtualizácie [18, 13, 3], ktorá tvorí základné jadro celej idey cloud. Primárnym zdrojom poznatkov v tejto kapitole bola kniha [17].

3.1 Úvod do cloud computingu

Výpočtová technika prechádza v súčasnosti transformáciou na model pozostávajúci zo služieb ponúkanych zákazníkom, kde zákazník pristupuje k týmto službám založených na ich presných požiadavkoch bez ohľadu na to, kde sú služby dostupné a akým spôsobom sú tieto služby zákazníkovi doručené. Existuje niekoľko paradigmát dávajúcich si za úlohu realizáciu spomínaného modelu služieb, vo svojej práci sa však primárne zameriam na *cloud computing*.
V základe sa jedná o infraštruktúru umožňujúcu klientom pristupovať k aplikáciám umiestneným na inom mieste než na lokálnom počítači, prípadne inom zariadení s internetovým pripojením. V praxi sa najčastejšie jedná o vzdialené datacentrá. Najvýraznejším prínosom cloud computingu je nepochybne eliminácia potreby zakúpenia sofvérových licencií pre celý podnik, nasadenia softvéru na lokálne zariadenia a v neposlednom rade aj eliminácia nákladov za obstaranie a správu serverov. Klient teda platí za využívanie aplikácie (prípadne za sadu aplikácií), pričom o správu, aktualizáciu softvéru a náklady za prevádzku serverov sa stará spoločnosť zaoberajúca sa prenájmom spomínaných služieb, tzv. hosting (obrázok 3.1). Výška poplatkov klienta voči týmto spoločnostiam sa odvíja hlavne od intenzity, s akou klient dohodnuté služby využíva - dôležitú úlohu hrá aj počet lokálnych zariadení v podniku, ktoré budú vzdialene k službám pristupovať.

Napriek významnému prínosu *cloud computingu*, sú s touto technológiou spojené aj určité negatíva. Jedným z nich je výpadok internetového pripojenia, prípadne iné problémy spojené s internetom na strane poskytovateľa (ISP), ktoré dnes už nie sú síce časté, avšak stále sú ovplyvňujúcim faktorom pri prístupe k vzdialeným službám.
Počas takéhoto výpadku by v ideálnom prípade prišlo iba k obmedzeniu, či spomaleniu činnosti zamestnancov podniku, v opačnom prípade sa môže jednať o kritickejšie následky. Výpadok však môže nastať aj na strane poskytovateľa hostingu. Je známy prípad z júla 2008, kedy nastal výpadok u Amazon S3 cloud servera druhýkrát v priebehu roka a služby

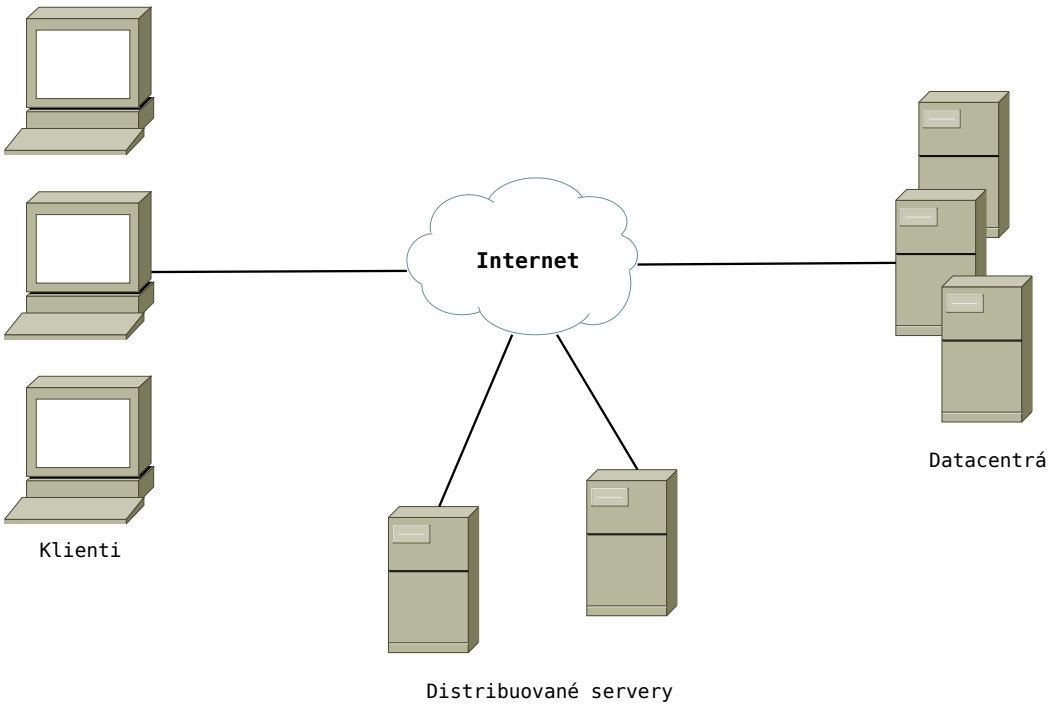


Obrázek 3.1: Hosting aplikácií špecializovanými spoločnosťami [17]

neboli klientom dostupné osem hodín.
Potenciálnym problémom by tiež mohla byť integrácia jednotlivých aplikácií, v prípade, že sú z geografického hľadiska na rozdielnych miestach. Príkladom je situácia, ak musí dôjsť k vzájomnej výmene dát medzi dvoma službami. Pokiaľ je totiž jedna služba dostupná lokálne a druhá vzdialene cez cloud, vzájomná komunikácia je komplikovanejšia, a zároveň oveľa náchylnejšia na chyby.
S cloud computingom sa často mýli pojem *grid computingu*. Internetové zdroje sa v názoroch na spoločné znaky cloud computingu a grid computingu líšia, ale podľa [6] majú tieto dve paradigmy spoločnú víziu, ktorou je redukcia nákladov za správu výpočtovej techniky, zvýšenie spoľahlivosti a flexibility tým, že všetky povinnosti a náklady spojené s výpočtovou technikou pripadnú na tretiu stranu.
Grid definuje a poskytuje sadu protokolov, nástrojov, middleware a služieb postavených nad týmito protokolmi. Tu nastáva problém s kompatibilitou medzi dvoma nezávislými grid systémami, nakoľko každý z nich disponuje unikátnou sadou protokolov. Naproti tomu cloud sa zameriava na súčasný beh niekoľkých menších aplikácií a dostupnosť poskytovaných služieb cez internet. Cloudy sú postavené na štandardných protokoloch pre webové služby a technológie, avšak je tu aj možnosť implementácie infraštruktúry cloudu nad existujúcimi grid technológiami. Podrobnejšie porovnania z rôznych perspektív od architektúry až po bezpečnostný, business, programový, dátový a výpočtový model, ako aj ďalšie vlastnosti, ktoré úzko spájajú alebo naopak odlišujú cloud od grid computingu, sú dostupné v článku [6].

3.2 Cloud komponenty

Predchádzajúca časť slúžila ako stručný úvod do architektúry cloud computingu. Na obrázku 3.2 sú znázornené tri základné elementy, ktoré z topologického hľadiska tvoria riešenie v podobe *cloud computingu*. Jedná sa o klientov (zariadenia), datacentrá a distribuované servery, pričom každá z týchto komponent plní špecifickú rolu v zmysle sprístupňovania cloud služieb zákazníkovi a bližšie budú tieto komponenty spomenuté v ďalších odstavcoch.



Obrázek 3.2: Komponenty znázorňujúce riešenie pomocou cloud computingu [17]

Z pohľadu cloud computingu sú ako klienti ponímaní, rovnako ako v prípade sieťovej komunikácie, stolné počítače, prípadne laptopy, tablety, mobilné telefóny a iné prenosné zariadenia. V skratke možno klientov popísať ako skupinu zariadení, s ktorými užívatelia interagujú, aby tak spravovali informácie a dáta uložené na cloud serveri. Pre lepší prehľad sú klientské zariadenia roztriedené do nasledujúcich kategórií:

- **Mobilné zariadenia** - Skupina zariadení zahŕňajúca PDA a chytré zariadenia (angl. *smartphone*).
- **Tenký (angl. *thin*) klienti** - Počítače, ktoré nedisponujú interným pevným diskom a všetky výpočtové operácie sa dejú na serveri. V praxi slúžia tieto zariadenia iba na zobrazovanie informácií dostupných zo serveru. Riešenie v podobe tenkých klientov sa stáva stále populárnejšie vzhľadom k výrazne nižšej cene oproti klasickým počítačom a nižšej energetickej spotrebe. Taktiež majú tieto počítače dlhšiu časovú periódu prevádzky, než je potrebné vymeniť hardvér (jednoduchá výmena zariadenia), prípadne sa zariadenia stanú úplne zastaralé. Medzi ďalšie benefity poskytované tenkými klientmi patrí určite bezpečnosť samotných zariadení. Ako bolo spomínané vyššie, zariadenia neobsahujú žiaden pevný disk, netreba v tomto prípade uvažovať hrozbu v podobe škodlivého softvéru. Zároveň sa všetky operácie vykonávajú na vzdialenom serveri, preto sú zariadenia v prípade krádeže prakticky nepoužiteľné.
- **Hrubý (angl. *thick*) klienti** - Klasické počítače využívajúce internetový prehliadač pri prístupe ku cloud službám.

Datacentrum chápeme ako kolekciu serverov, na ktorých je nasadená daná aplikácia, prípadne sada aplikácií. Datacentrá sú umiestňované do špeciálnych miestností - serverovní,

v podstate kdekoľvek na svete, kde sú dáta dostupné cez internet. Rastúcim trendom sa však stáva virtualizácia serverov. Vďaka virtualizačnej vrstve je možné na jednom fyzickom serveri spustiť viac inštancií virtuálnych serverov s rôznymi operačnými systémami, pričom všetky inštancie sú vzájomne izolované. Počet virtuálnych strojov bežiacich na jednom fyzickom serveri závisí od veľkosti jeho úložného miesta, dostupného výkonu a taktiež na aplikáciách a ich nárokoch na výpočtové zdroje. Virtualizáciou sa dosiahne výrazného zníženia nákladov na napájanie serverov a tiež ich údržbu.

Anthony T.Velte v [17] k distribuovaným serverom uvádza že pre servery platí, že nemusia byť uložené na rovnakom mieste. V praxi sa často súvisiace servery nachádzajú na geograficky odlišných miestach. Z pohľadu zákazníka a užívateľa služieb cloudu sa servery zdajú byť na jednom mieste. Výhodou je, že v prípade nedostupnosti služby na jednom serveri prostredníctvom webovej stránky je služba často dostupná cez inú stránku a iný server. Tento model taktiež prináša výhody v údržbe serverov, pretože nové servery stačí pridať do niektorej zo serverovní, sprístupniť ich pomocou webovej stránky a integrovať ho do cloudu.

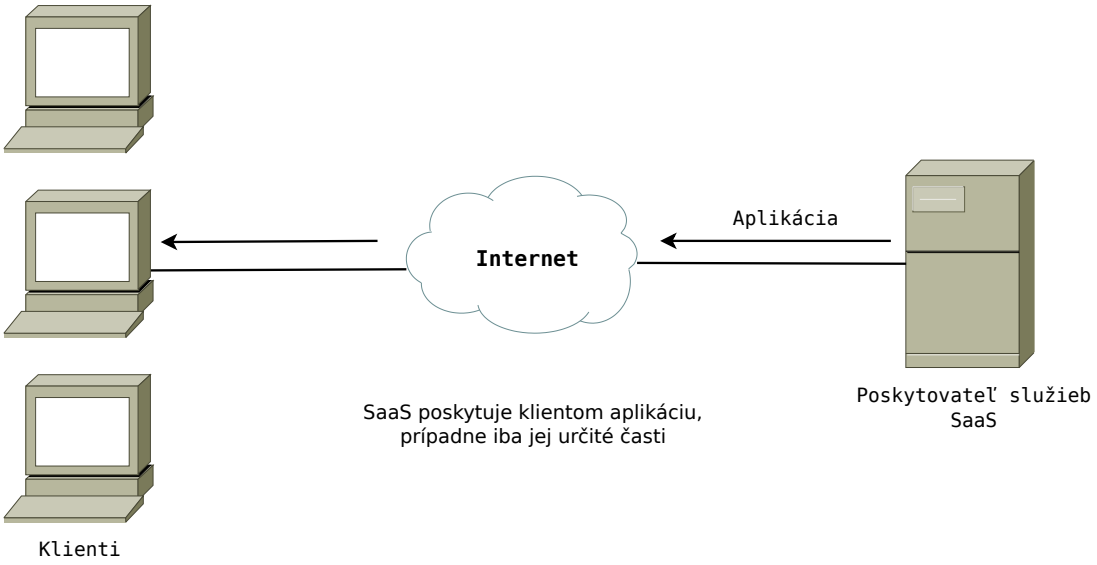
3.3 Služby

V tejto časti budú predstavené tri najrozšírenejšie modely, na základe ktorých poskytovatelia cloud computingu ponúkajú svoje služby. Sú to SaaS, PaaS a IaaS.

Software as a service

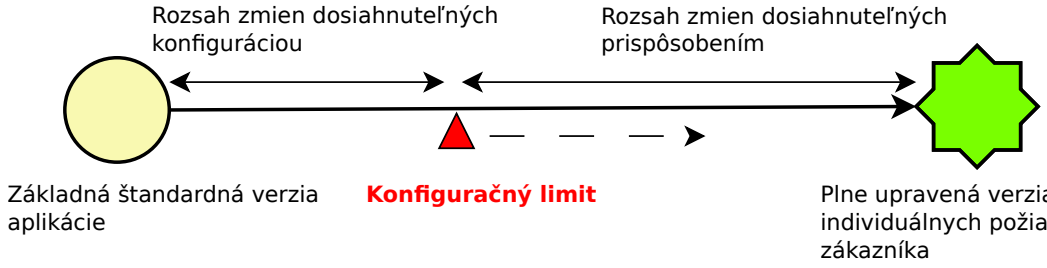
Model softvér ako služba (angl. *Software as a Service*, ďalej ako *SaaS*) sa vyznačuje doručovaním funkcionality softvéru veľkej skupine zákazníkov za využitia webových služieb, pričom samotná aplikácia existuje ako jediná inštancia bežiaca na multi-nájomnej (angl. *multi-tenancy*) platforme. Zákazníci potom jednoducho od poskytovateľa služieb obdržia autentizačné údaje a pristupujú k dohodnutým službám cez webový prehliadač. Jednoduchá ilustrácia tohto prístupu je znázornená na obrázku 3.3.

Každý klient však má na konkrétnu aplikáciu špecifické požiadavky a nie je možné pre všetky variácie požiadavkov vyvíjať separátnu verziu totožného produktu. V ideálnom prípade by bol každý klient spokojný so štandardnými nástrojmi a konfiguráciou dohodnutej služby. Realita a prax však ukazujú, že takýto prípad v oblasti podnikových aplikácií takmer nikdy nenastáva. Avšak aby bolo možné vyhovieť zákazníkom a ich individuálnym požiadavkom na softvér, používajú sa dva osvedčené prístupy, menovite ide o prispôsobenie a konfiguráciu. Fundamentálny rozdiel medzi týmito dvoma pojmami je miera komplexnosti úpravy pri napĺňaní individuálnych požiadavkov. Konfigurácia nezahŕňa žiadne úpravy zdrojového kódu SaaS aplikácie. Zvyčajne zahŕňa iba zmeny ako zmenu nastavenia preddefinovaných parametrov, pridávanie dátových položiek, vlastných tlačítok, zmenu názvov jednotlivých položiek, modifikácia roletového menu (angl. *drop-down menu*). Konfigurácia podporuje modifikáciu softvéru iba v rozsahu preddefinovaného konfiguračného limitu (obrázok 3.4).
Ďalšie úpravy funkcionality aplikácie na mieru požiadavkom už vyžadujú zásah do zdrojového kódu - prispôsobenie. V porovnaní s konfiguráciou, prispôsobenie je oveľa nákladnejšie riešenie ako pre poskytovateľa, tak pre zákazníka. Zvýšenie nákladov je primárne spôsobené nutnosťou spravovať jednotlivé verzie kódu zahŕňajúce dlhší životný cyklus aplikácie -



Obrázek 3.3: Software as a service [17]

vývoj, ladenie, testovanie, nasadenie. Z tohto dôvodu sa SaaS pri napĺňaní individuálnych požiadavkov snaží uprednostniť konfiguráciu pred prispôsobením. Táto sekcia si dáva za cieľ oboznámiť čitateľa so základnými princípmi a funkcionalitou cloudových technológií, preto je nadrámec práce sa ďalej podrobnejšie zaoberať konfiguračnými technikami a technikami pre prispôsobenie, pre bližšie informácie však možno nahliadnuť do [15]. Všeobecne však platí, že so zvyšujúcou sa funkčnou zložitosťou, je nutné potenciálne vynaložiť väčšie úsilie uspokojiť zákaznícke požiadavky.



Obrázek 3.4: Konfiguračný limit medzi konfiguráciou a prispôsobením [15]

Ako už bolo v kontexte cloud computingu spomínané vyššie, jeden z najväčších prínosov SaaS sú určite nižšie náklady v porovnaní s investíciou do licencií na lokálne počítače. Medzi ďalšie prínosy a výhody by sme mohli zaradiť nasledovné:

- **Familiárnosť s technológiou World Wide Web** - Väčšina pracovníkov má prístup na internet a je schopná používať webový prehliadač na prezeranie internetového obsahu. Krivka učenia externých aplikácií dostupných cez webový prehliadač je tým pádom menšia.
- **Prispôsobenie aplikácií** - SaaS aplikácie sú v porovnaní so staršími aplikáciami oveľa ľahšie modifikovateľné smerom k individuálnym požiadavkom zákazníka.

- **Spoľahlivosť webových technológií** - V predchádzajúcich častiach bol spomenutý fakt, že web je potenciálnym zdrojom nechcených a neočakávaných chýb a komplikácií. Tie však nie sú tak časté, a preto sa v globálnom merítku považuje za spoľahlivý.

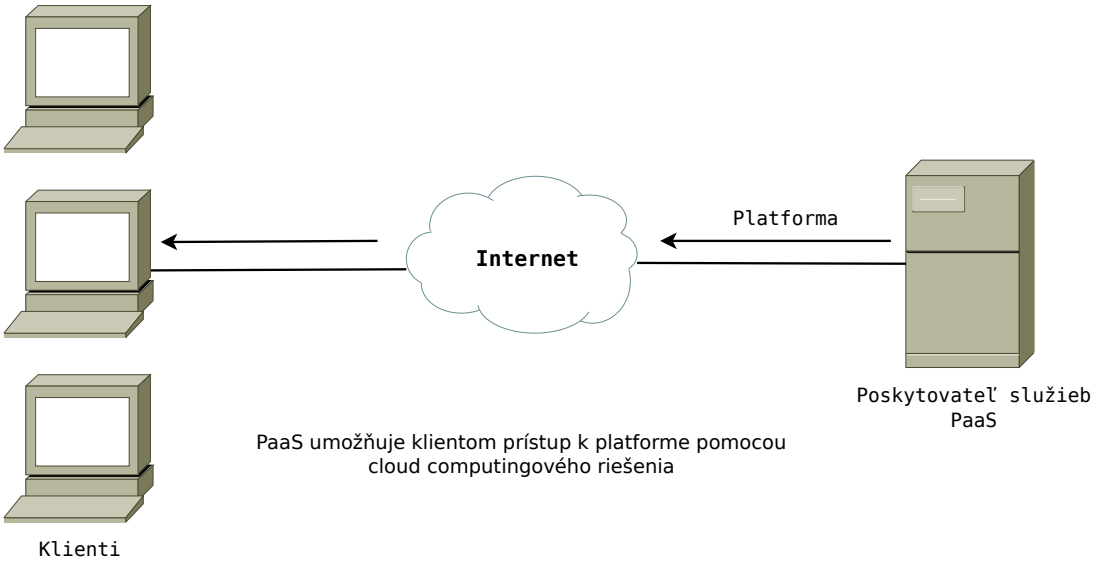
- **Väčšie prenosové pásmo** - Vďaka neustálej snahe o zvýšenie prenosového pásma a taktiež o vylepšenie kvality služieb QoS, môžeme pozorovať zvýšenie dátového toku a aj jeho spoľahlivosti. Dôsledkom toho dnes organizácie vkladajú do cloud technológií stále väčšiu dôveru, pričom požadujú nízke latencie a vysoké rýchlosti pri prístupe k externým službám.

Samozrejme aj SaaS čelí určitým prekážkam v jeho implementácií a použití. Prvou je situácia, ak konkrétna spoločnosť vzhľadom k svojim príliš špecifickým výpočtovým potrebám nie je schopná nájst vhodnú aplikáciu dostupnú cez SaaS, avšak napriek svojim individuálnym požiadavkom by táto spoločnosť mohla nájsť niektoré z potrebných komponent dostupné ako SaaS.
Ďalej je tu problém, ktorý sa v súvislosti s poskytovateľmi SaaS označuje ako *lock-in*. V princípe ide o to, ak zákazník zaplatí za využívanie konkrétnej služby poskytovateľovi, pritom uvažujme určitú úroveň konfigurácie či prispôsobenia, nemusí byť v budúcnosti možné túto aplikáciu preniesť k inému poskytovateľovi, prípadne ak to možné je, bude si predošlý poskytovateľ nárokovať vysoký poplatok za takýto prenos. Ďalšou výzvou pre SaaS by taktiež mohla byť rastúca dostupnosť a popularita opensource aplikácií spolu s klesajúcimi cenami hardware. V takom prípade by mohla spoločnosť výhodne nakúpiť dostatočný výpočetný výkon a nasadiť naň práve opensource aplikácie.

Platform as a service

V predošlej časti sme si popísali základné vlastnosti a princípy modelu SaaS, kde sa užívatelia nemusia starať o údržbu, aktualizáciu, hosting, uloženie dát a vývoj aplikácie. O to všetko sa stará poskytovateľ. Avšak niektorí poskytovatelia vzali koncept SaaS, doplnili ho a začali poskytovať tzv. platformu ako službu (angl. *Platform as a service*, ďalej len *PaaS*) systémy [8]. Ilustrácia topológie modelu PaaS, od ktorej sa v ďalšom texte odrazíme, je znázornená na obrázku 3.5.

PaaS systémy sú rovnako ako SaaS hostované platformy, avšak primárne určené na online vývoj webom dostupných aplikácií. Platformy poskytujú všetky nevyhnutné zdroje vrátane end-to-end prostredia pre vývoj aplikácií a služieb cez internet, pričom služby PaaS zahŕňajú dizajn aplikácie, vývoj, testovanie, nasadenie a samozrejme hostovanie. Taktiež hodno spomenúť integráciu webovej služby, integráciu databázových systémov, bezpečnosť, škálovateľnosť a verzovanie. V niektorých prípadoch môžu vývojári využívať online zdroje poskytovateľa PaaS a vyvíjať aplikácie offline. Často je tiež možné lokálne do vyrovnávacej pamäti cache uložiť časť programu a pracovať offline. Pri nasledujúcom pripojení k online aplikácií dôjde automaticky k synchronizácii vykonaných zmien.
Poskytovatelia PaaS taktiež mnohokrát usilujú o uľahčenie práce vývojárom podporou obľúbených a rozšírených programovacích jazykov ako C, Java, PHP, prípadne špecializovaných drag-and-drop nástrojov, ktoré implementujú príslušné bloky kódu, a tým uľahčujú prácu menej skúseným vývojárom. Niektorí poskytovatelia dokonca vytvorili vlastné špecializované programovacie jazyky, ktoré by podľa slov samotných tvorcov mali proces vývoja výrazne urýchľovať, zároveň by mali byť relatívne ľahké na naučenie.
Miernym obmedzením, ktoré PaaS so sebou prináša je, že poskytovateľ určuje infraštruktúru samotnej aplikácie, teda na akom druhu operačného systému aplikácia pobeží, použité



Obrázek 3.5: Platform as a service [17]

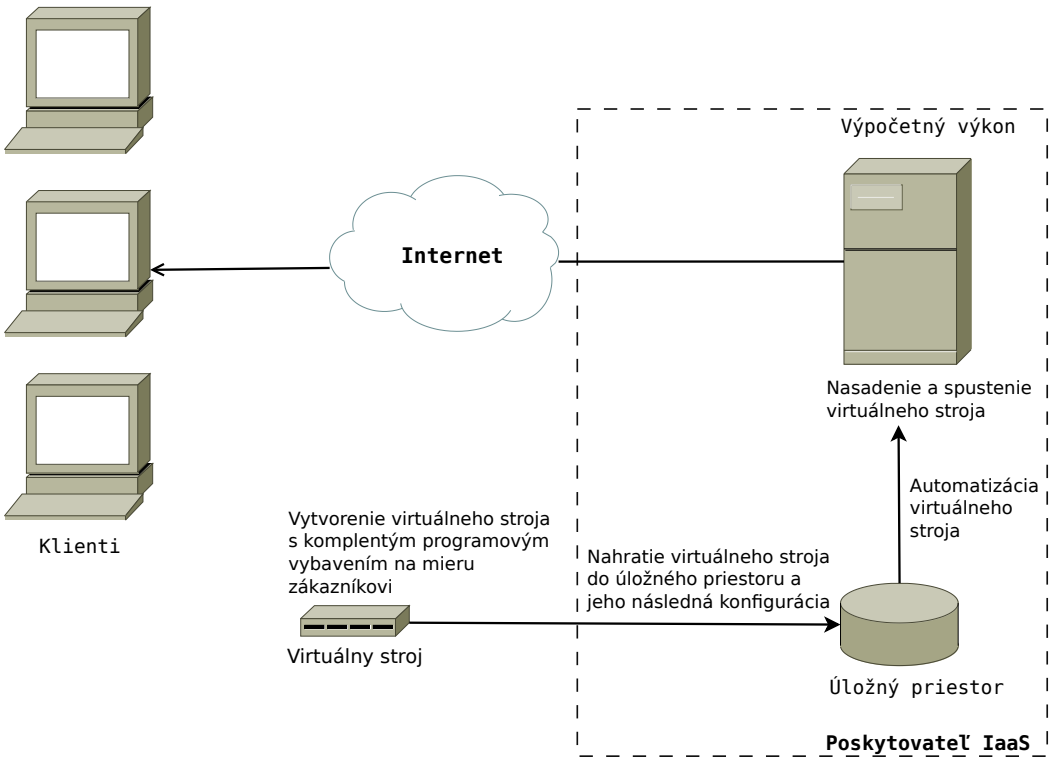
API, programovací jazyk, ale aj možnosti následnej správy aplikácie. Užívatelia teda pri vývoji používajú výhradne nástroje a prostredie dostupné od poskytovateľa. PaaS vytvára virtuálnu platformu na vývoj a nasadenie. Samotný vývoj prebieha interakciou so serverami poskytovateľa. Pri vývoji sa programátor zaoberá výhradne problémami súvisiacimi s programovým tokom a aplikačnou logikou, pričom fyzický systém a sieťové rozhranie nie je vôbec nutné uvažovať. Dôvodom je virtualizácia serverov, ktorá je úzko spätá s technológiou cloud computingu a je jej venovaná sekcia 3.4. PaaS vďaka nej úspešne skrýva komplexnosť logiky medzi klientom a virtuálnym serverom pomocou tzv. virtualizovanej infraštruktúry.

V nasledujúcom odstavci si stručne vymenujeme klady, ale aj nedostatky tohto modelu. G. Lawton v [8] uvádza, že podľa zástancov PaaS hostovanie celého vývojového prostredia zvyšuje produktivitu programátorov a dovoľuje spoločnostiam vydať produkt oveľa rýchlejšie pri nižších nákladov. Prístup PaaS eliminuje potrebu vývojárov konfigurovať si vlastné servery za účelom vývoja a testovania aplikácií, škálovať si vývojové prostredie, či implementovať a integrovať si nástroje na správu. V neposlednom rade je tu virtualizácia, ktorá ich odbremení od práce so sieťovými rozhraniami, úskaliami konkrétneho operačného systému a aktualizáciou serverových systémov. Naopak z pohľadu záporov, najväčším nedostatkom naďalej zostáva podobne ako u SaaS prenositeľnosť medzi jednotlivými poskytovateľmi.
Tým, že poskytovatelia založili svoje sytémy na proprietárnych službách a často aj na vlastných programovacích jazykov, pre vývojárov môže nastať už spomínaný „lock-in“ na konkrétneho poskytovateľa. Ďalšími nepríjemnými situáciami sú výpadok internetu, a tým strata aktuálnych neuložených dát alebo stavu aplikácie, a potenciálny krach poskytovateľa PaaS služieb a spolu s ním aj strata všetkých dát zákazníka.

Infrastructure as a service

Infraštruktúra ako služba (angl. *Infrastructure as a service*) je posledným z trojice najčastešie používaných modelov cloud computingu. Poskytovatelia IaaS služieb ponúkajú

vypočetný výkon serverov spolu s úložným priestorom a sieťovým rozhraním (obrázok 3.6). V kontexte IaaS sa objavuje ďalší podobný model, *Hardware as a service*. Niektoré literatúry tieto modely striktne rozlišujú, avšak Antony T. Velte v [17] dáva IaaS a HaaS do synonymického vzťahu.
Z pohľadu zákazníka sa však v rámci IaaS jedná o určitý dohodnutý počet virtuálnych strojov (počet sa odvíja od výkonnostných požiadavkov zákazníka), na ktoré sa nasadí zákazníkov operačný systém (prípadne viacero rôznych OS) spolu s aplikáciami, o ktoré má záujem, aby na serveri bežali. Toto riešenie je obzvlášť výhodné, pokiaľ by bolo časovo náročné a finančne nákladné upraviť kód existujúceho softvéru tak, aby spĺňal požiadavky modelov SaaS či PaaS. Sushil Bhardwaj vo svojom článku [1] uvádza, že narozdiel od modelov PaaS a SaaS, sa poskytovateľ služieb IaaS stará výhradne o správu, údržbu a napájanie datacentier. Všetky ostatné povinnosti, menovite vývoj, správa a nasadzovanie aplikácií pripadajú na zákazníka, ako keby pracoval s vlastnými datacentrami. Opäť je výhodou forma, ktorou zákazník platí za prenájom dohodnutých služieb, pretože platí iba za výkon, ktorý využíva. Výhodou tohto prístupu je flexibilita, kedykoľvek je možné požiadavky na výkon rozšíriť (za adekvátne navýšenie sumy), prípadne pri dočasnej potrebe vyššieho výkonu je možné po uplynutí danej periódy znížiť požiadavky na výkon u poskytovateľa a s tým spojené náklady.



Obrázek 3.6: Infrastructure as a service [17]

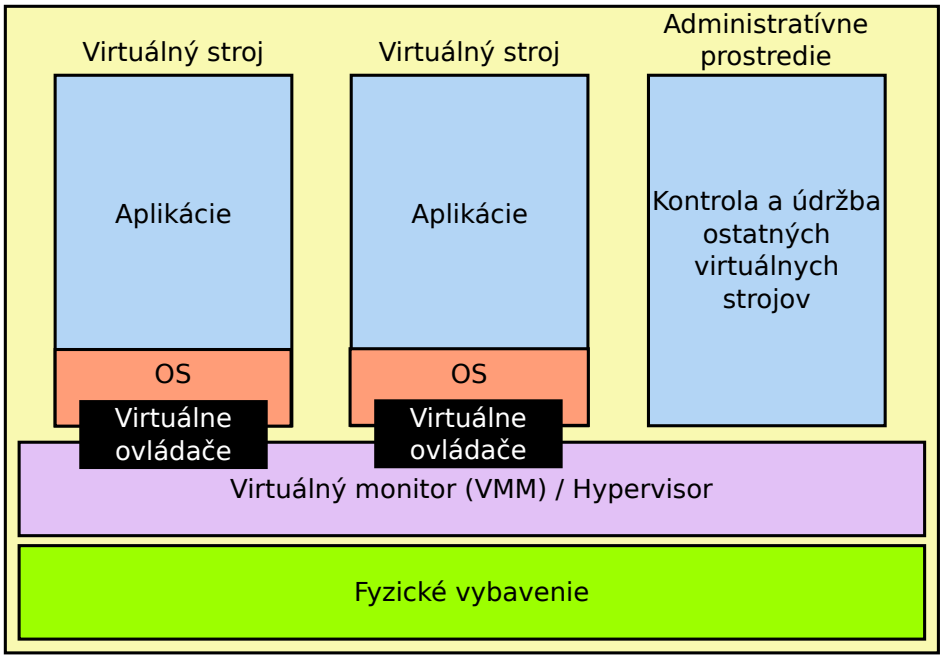
Na záver tejto časti venovanej jednotlivým modelom cloud computingu uvádzam krátky bodový prehľad požiadavkov, ktoré sú zo strany zákazníka vždy kladené na cloud computing:

- dostupnosť služby/aplikácie odkiaľkoľvek s prístupom na internet
- flexibilný, škálovateľný, virtualizovaný systém

- odolnosť a prístupnosť aplikácie (na začiatku kapitoly 3 bol spomenutý problém výpadku internetového spojenia, a tiež porucha fyzických zariadení)
- menšie náklady v porovnaní s nákladmi na údržbu a napájanie datacentier a nákup licencií krabicového softvéru
- povinnosti poskytovateľa na údržbu datacentier, údržbu a aktualizáciu aplikácie (SaaS), dátovú bezpečnosť (SaaS)
- jednoduchý a jednotný prístup k službe cez internetový prehliadač

3.4 Virtualizácia

Virtualizácia je jedným z najdôležitejších elementov celého cloud computingu. Ide o technológiu pomáhajúcu IT organizáciám optimalizovať výkon aplikácií nákladovo efektívnym spôsobom. Základom je virtuálny stroj, ktorý sa správa ako fyzický počítač, beží na ňom operačný systém spolu s aplikáciami, avšak jeho úlohou je odtieniť skutočný použitý hardvér od zákazníka. Táto sekcia čerpá poznatky nadobudnuté z [13, 9, 3, 18].
Medzi najpoužívanejšie virtualizačné techniky, ktoré dovoľujú súčasný beh viacerých operačných systémov, angl. *guests*, na jednom fyzickom počítači, patrí technika *hypervisor*. Hypervisor poskytuje hostovaným operačným systémom virtuálnu platformu, a zároveň monitoruje beh systémov na týchto platformách.
Všeobecne platí, že hypervisor je nainštalovaný na serveri a jeho jedinou úlohou je zabezpečiť beh hostovaných, na sebe nezávislých, operačných systémov, pričom hypervisor im poskytuje rovnakú sadu generických virtuálnych ovládačov. Vďaka tejto sade virtuálnych ovládačov, ktoré hypervisor už v sebe zahŕňa, je možný simultánny beh rôznych operačných systémov (obrázok 3.7).

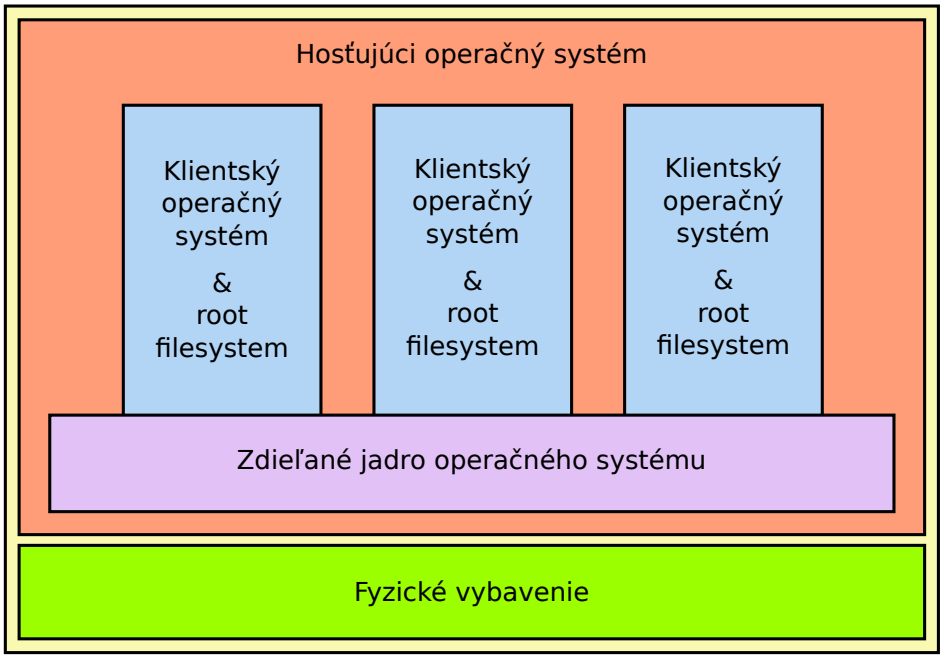


Obrázek 3.7: Plná virtualizácia s virtuálnym monitorom, tzv. hypervisor [13]

Hypervisor však vyžaduje na svoj beh procesorový čas. V praxi to znamená, že fyzický server musí pre beh inštancie hypervisor rezervovať dostatočné prostriedky, čo sa môže negatívne prejaviť na celkovom výkone serveru a rýchlosti jednotlivých aplikácií hostovaného operačného systému. Rozlišujeme tri základné úrovne virtualizácie: plná virtualizácia, paravirtualizácia a virtualizácia na úrovni OS.
Pri plnej virtualizácii sa podľa prof.Matysku v [9] jedná o dôsledné virtualizovanie všetkých súčastí počítača. Ponúkame teda prostredie, v ktorom operačný systém nemôže žiadnym spôsobom poznať, že nemá prístup k fyzickému technickému vybaveniu. Operačný systém ani aplikácie nepotrebujú žiadne modifikácie. Môžeme v podstate hovoriť o ideálnom stave, kedy dochádza k plnému oddeleniu fyzickej vrstvy, všetky programy bežia výhradne na virtuálnom hardvéri a prístup k fyzickému vybaveniu je vždy sprostredkovaný hypervisorom. U plnej virtualizácie nemusí existovať žiadna jednoduchá väzba medzi virtuálnym prostredím a konkrétnym hardvérom, na ktorom je virtuálny počítač prevádzkovaný. Tým je umožnená plná prenositeľnosť.
Plná virtualizácia má však svoju cenu. Vzhľadom k tomu, že dochádza k úplnému oddeleniu fyzickej a programovej vrstvy, je pri plnej virtualizácii prakticky nemožné dosiahnuť plného výkonu i v prípade, ak je virtuálny počítač viacmenej presným obrazom hardvéru, na ktorom beží. Hypervisor totiž musí kompletne odtieniť virtuálny počítač od akejkoľvek zmeny hardvéru tým, že emuluje fyzické vybavenie a väčšinu operácií vykonáva vo vlastnom softvéri namiesto toho, aby ich vykonával hardvér.

Paravirtualizácia sa naopak vyznačuje tým, že uskutočňuje iba čiastočnú abstrakciu na úrovni virtuálneho počitaču, t.j. ponúka virtuálne prostredie, ktoré je podobné tomu fyzickému. Virtualizácia v tomto prípade nie je úplná a niektoré vlastnosti procesoru môžu byť obmedzené a operačný systém dokáže rozpoznať, že beží vo virtuálnom prostredí. Na druhú stranu skutočnosť, že virtuálny a fyzický hardvér sa príliš nelíšia, umožňuje, aby virtuálny počítač využíval vlastnosti fyzického prostredia v maximálnej miere (nemusíme emulovať všetky komponenty virtuálneho počítaču).
Narozdiel od plnej virtualizácie, kde operačný systém nemal žiadnu informáciu o tom, že je virtualizovaný, pri paravirtualizácii je snaha o čo najefektívnejšiu komunikáciu medzi klientským operačným systémom a virtuálnym monitorom. Je však na to nutná zmena jadra klientského operačného systému zámenou kritických systémových volaní za volania virtuálneho monitoru (angl. *hypercalls*), pomocou ktorých klientský operačný systém priamo komunikuje s virtualizačnou vrstvou [18]. Vzhľadom k tomu, že paravirtualizácia nedokáže pracovať s nemodifikovanými operačnými systémami, problémom môže byť kompatibilita a portabilita. Z tohto dôvodu boli vyvinuté špecializované systémy postavené na paravirtualizáciu, z ktorých najznámejšie sú *VMWare workstation* a *Xen*.

Posledný prístup umožňuje virtualizáciu na úrovni hosťujúceho operačného systému, ktorý nahrádza úlohu virtuálneho monitoru (hypervisor). Z obrázku 3.8 je vidno, že hosťujúci operačný systém podporuje viacero izolovaných virtualizovaných operačných systémov na jednom fyzickom serveri, pričom tento hosťujúci systém má ako jediný plnú kontrolu nad virtualizovanými systémami a infraštruktúrou hardvéru. Tento prístup je síce zo všetkých najjednoduchší, avšak za cenu väčšej zraniteľnosti. Príkladom môže byť útok na samotný hosťujúci operačný systém, pomocou ktorého je útočník schopný ovládnuť všetky virtuálne stroje.



Obrázek 3.8: Virtualizácia na úrovni operačného systému [13]

Kapitola 4

Návrh a implementácia aplikácie

V tejto kapitole je popísaný návrh a implementácia jednotlivých častí aplikácie. Prvá časť je zameraná na návrh užívateľského rozhrania a samotnej klientskej časti aplikácie. V nasledujúcej sekcii 4.2 je popísaný návrh a implementácia servrovej časti aplikácie, databázový model a nasadenie tejto webovej aplikácie ako cloud službu typu SaaS na platformu Openshift [12]. V poslednej sekcii tejto kapitoly je popísaný návrh protokolu a integrácia funkcie streamovaného prehrávania do grafickej časti zo sekcie 4.1.

4.1 Návrh a implementácia užívateľského rozhrania

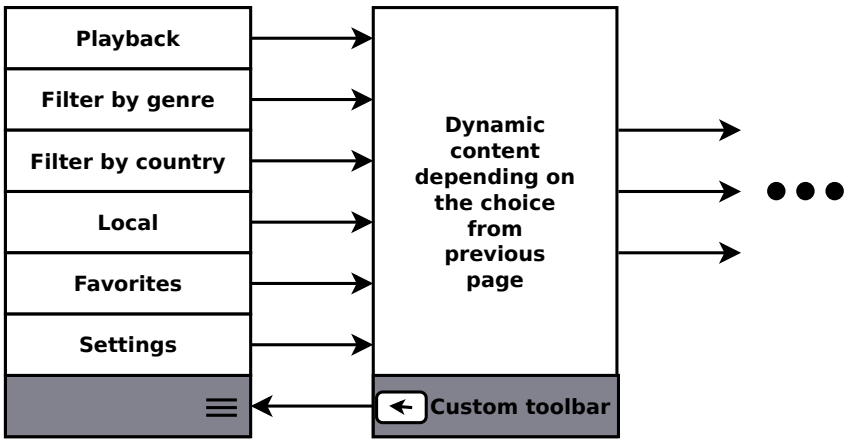
Použité nástroje

Prvým bodom návrhu bolo zvolenie cieľovej platformy. Tu bola zvolená Nokia N9, avšak pre účely testovania bola použitá jej upravená verzia určená pre vývojárov s označením N950 s operačným systémom MeeGo, ktorá prichádza s možnosťou inštalácie špeciálneho vývojárskeho balíku, zahŕňajúceho množstvo nástrojov určených pre ladenie aplikácií, sieťovú komunikáciu, analýzu spotreby a iné. Pre základné testovanie užívateľského rozhrania bol využitý simulátor dostupný ako súčasť vývojového balíku Qt SDK 1.2.1. Pre finálne testovanie aplikácie, zahŕňajúce prehrávanie multimediálneho obsahu, bolo použité fyzické zariadenie.
Ako bolo spomenuté v predchádzajúcom odstavci, pre vývoj aplikácie bola použitá sada knižníc (angl. *framework*) Qt. Z pohľadu grafického užívateľského rozhrania sa so zariadeniami Nokia spája balík Qt Quick, ktorého súčasťou je sada elementov, aplikačné rozhrania C++ pre integráciu elementov s klasickými Qt aplikáciami a v neposlednom rade aj deklaratívny skriptovací jazyk QML (angl. *Qt Modeling Language*). Jazyk vo svojej podstate vychádza z Jazyka JavaScript a je určený práve pre návrh užívateľských rozhraní, primárne mobilných aplikácií, kde medzi kľúčové faktory patrí dotykové rozhranie a rôzne druhy animácií. Jeho sila spočíva v používaní preň typických stavebných blokov, nazývaných elementy. Jedná sa zväčša o grafické prvky (*Rectangle, Image, ToolIcon, atď.*), ale spadajú sem i behaviorálne prvky (*State, Transition*). Tieto prvky prinášajú vyššiu úroveň abstrakcie a intuitívnosti v ich použití, pričom samy sa opierajú o silu a efektivitu knižníc Qt. Čo sa ich používania týka, elementy disponujú rôznymi atribútmi a premennými, ktoré pomáhajú lepšie definovať ich výzor, umiestnenie, prípadne správanie. V ďalšom odstavci budú uvedené konkrétne použitia týchto elementov.

Návrh

Pri návrhu grafického užívateľského rozhrania bola snaha uplatniť a dodržať základné ustálené kritériá, teda intuitívnosť používania a rýchlosť prístupu ku kľúčovým položkám aplikácie. Ďalším dôležitým požiadavkom na rozhranie bolo sledovanie zaužívaných metód, uplatňovaných pri návrhu aplikácie na mobilné zariadenia Nokia (angl. *Design guidelines*). To znamená používanie vstavaných tlačidiel, ikon, jednotný vzhľad panelu nástrojov (ďalej len *Toolbar*) a jeho štandardné umiestnenie na spodu obrazovky, používanie systému nasúvania stránok pri vynútenej zmene obsahu obrazovky, a iné. Návrh užívateľského rozhrania je uvedený na obrázku 4.1. Obrázok už zahŕňa aj systém prepínania jednotlivých stránok (reakcia na kliknutie je v znázornená oranžovo). Rozhranie bolo logicky rozčlenené na štyri logické celky (*Prehrávanie, Prehliadanie, Obľúbené, Nastavenia*).

Za povšimnutie stojí najmä element *Toolbar*, ktorý sa svojím jednotným výzorom na všetkých stránkach rozchádza s tradičným použitím, kde takmer každá stránka disponuje samostatnou sadou nástrojov, ktoré sa zobrazujú na tomto paneli. V prvotnom návrhu bola uvažovaná hlavná stránka, ktorá by plnila úlohu akéhosi rozcestníku, kde by sa ďalšie stránky zobrazovali na základe aktuálne zvolenej položky v zozname. Tým pádom by každá stránka mohla mať vlastnú sadu nástrojov, pretože práve hlavná stránka by určovala nasledujúcu stránku. Avšak každá stránka, prípadne sada stránok, ak hovoríme o delení do logických celkov, by musela obsahovať tlačidlo späť na hlavný rozcestník a odtiaľ ďalej pokračovať na práve požadovanú stránku (obrázok 4.2).

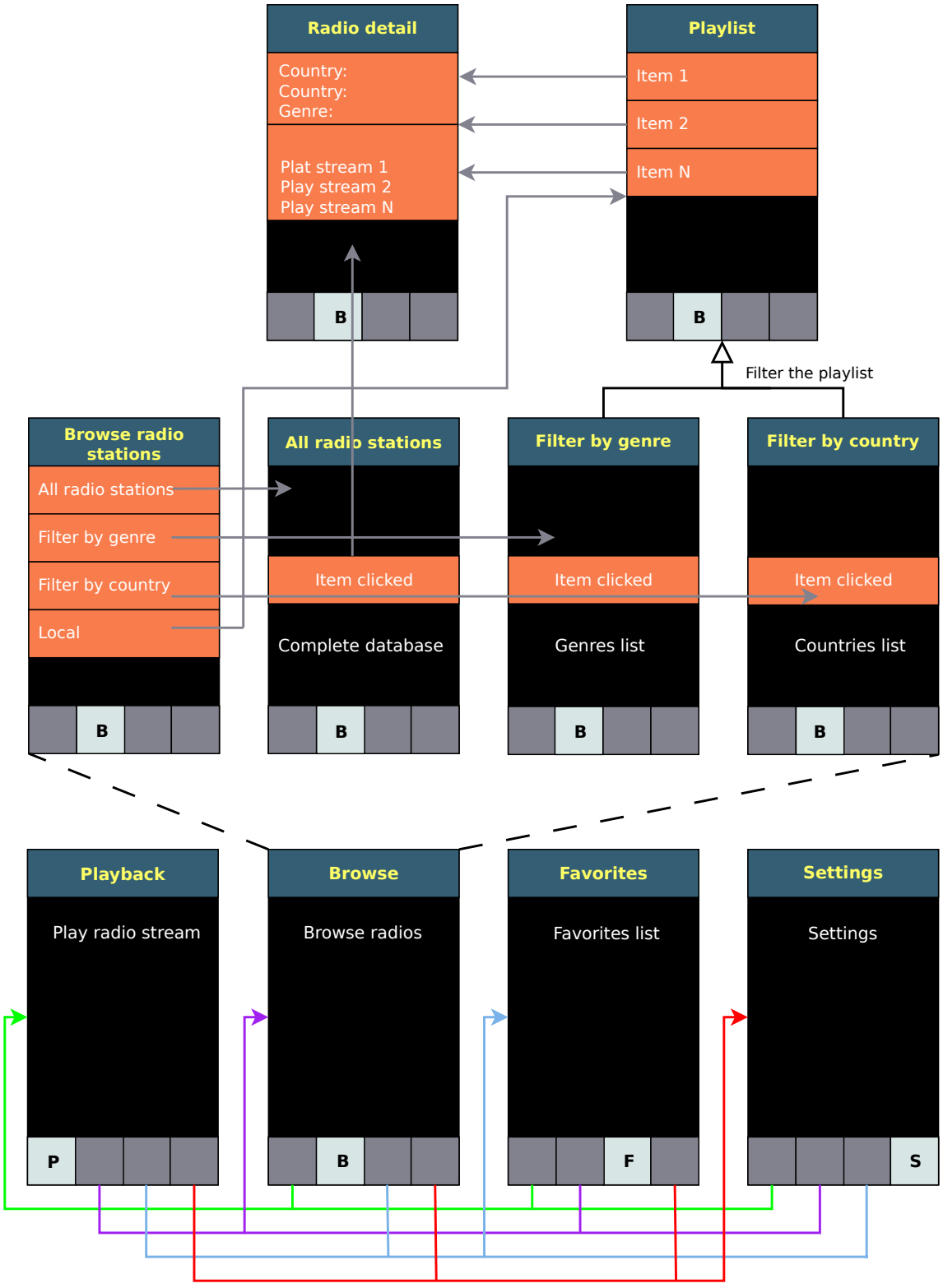


Obrázek 4.2: Návrh užívateľského rozhrania s hlavným rozcestníkom

Aktuálny návrh z obrázku 4.1 si kladie niekoľko cieľov:

- rýchlosť ovládania
- efektívny prístup ku kľúčovým položkám aplikácie
- prehľadnosť užívateľského rozhrania

Prvé dva sú veľmi úzko previazané. V praxi to znamená, že užívateľ by mal byť schopný intuitívne odhadnúť smerovanie svojho požiadavku do správneho logického celku a v ideálnom prípade by rozhranie malo vyhovieť jeho požiadavku na jediné kliknutie. Toto je však nie vždy možné, ako bude spomenuté v neskorších odstavcoch. Napriek tomu k dosiahnutiu týchto cieľov výrazným spôsobom prispieva jednotný vzhľad panelu nástrojov na všetkých



Obrázek 4.1: Diagram prepínania stránok zvoleného návrhu

stránkach. Toolbar pracuje na princípe prepínania kariet, pričom navzájom prepája všetky štyri spomínané logické celky. Zároveň sa ním eliminovala potreba zvláštnej stránky - rozcestníku.
Posledným spomínaným cieľom je prehľadnosť rozhrania, to značí, že užívateľ by nikdy nemal nadobudnúť pocit, že sa v aplikácií stratil, vždy musí byť informovaný o každej akcii, ktorú vykoná, prípadne musí byť vyzvaný na potvrdenie požadovanej operácie. K tomu slúžia dialógové okná, vhodná voľba ikon (tzn. ikona je dostatočne intuitívna, nepotrebuje žiaden ďalší popis, ktorý by definoval správanie aplikácie skryté za touto ikonou). V neposlednom rade bola zvolená vrchná lišta, na ktorej sa vždy zobrazujú názvy odpovedajúcich logických celkov, resp. podcelkov danej stránky, napr. *Nastavenia, Filtrovanie rádií, Filtrovanie podľa XY, Detail rádia UV* alebo iné doplňujúce informácie.
Základom každého projektu v Qt Quick je hlavný súbor main.qml reprezentujúci hlavné okno a súbor MainPage.qml obsahujúci všeobecné nastavenia vzhľadu stránok, nastavenie počiatočnej stránky, prípadne je možné v tomto súbore všetky ďalšie použité stránky predom deklarovať.
Stručný prehľad jednotlivých stránok k obrázku 4.1:

- **Prehrávanie**
  - hlavná stránka
  - informácie o práve prehrávanom rádiu a aktuálnej prenosovej rýchlosti
  - možnosť pridania aktuálneho streamu do zoznamu obľúbených
- **Prehliadanie**
  - prehliadanie rádií
  - filtrovanie zoznamu rádií
  - vyhľadávanie v zozname rádií
  - náhľad na detail zvoleného rádia
- **Obľúbené**
  - zoznam obľúbených streamov konkrétnych rádií
  - po kliknutí zahájenie prehrávania a prechod na stránku prehrávania
  - po pridržaní voľby vysunutie kontextového menu s možnosťou odstránenia zvoleného streamu zo zoznamu alebo jeho nahlásenie z dôvodu nefunkčnosti.
- **Nastavenia**
  - nastavenia témy a možnosti zahájiť prehrávanie ihneď po štarte aplikácie
  - zahŕňa odkaz na stránku venovanú popisu aplikácie

Implementácia

V tomto odstavci budú bližšie popísané črty filtrovanie a vyhľadávanie rádií.
Po aktivácii stránky *Prehliadanie* sa zobrazí zoznam aplikovateľných filtrov, konkrétne *Všetky rádiá, Filtrovanie podľa žánrov, Filtrovanie podľa krajín* a *Miestne*. Pre každú kategóriu filtrov je dedikovaný samostatný element *ListModel*, ktorý je naplnený v okamihu úspešného prevzatia dát z databázy (viac o návrhu API a prehrávaní v sekcii 4.3).

Po zvolení špecifického filtru (napr. rádia v Českej Republike) dôjde k volaniu funkcie `applyLocalFilter()`, ktorá z pôvodného komplexného zoznamu rádií vyberie iba rádiá s príznakom zvolenej krajiny a vloží toto rádio do filtrovacieho modelu, ktorý sa po ukončení funkcie nastaví ako zdroj dát pre stránku zobrazenia aktuálneho zoznamu[1].
Spomínaná položka *Miestne* je technicky súčasťou filtrovania podľa krajín, avšak z dôvodu efektívnejšieho prístupu k miestnym rádiám bola umiestnená práve na koreňovú stránku prehliadania. Zoznam miestnych rádií je vytvorený na základe výsledku IP geolokalizácie, kedy sa zariadenie dotazuje metódou `GET http` protokolu na špecializovaný server, ktorý na základe IP adresy určí umiestnenie zariadenia a do tela odpovede priloží serializované dáta v notácii JSON a v nasledovnom formáte:

```
"ip":"147.229.186.100",
"country_code":"CZ",
"country_name":"Czech Republic",
"region_code":"78",
"region_name":"Jihomoravsky kraj",
"city":"Brno",
"zipcode":"",
"latitude":49.2,
"longitude":16.6333,
"metro_code":"",
"areacode":""
```

Příklad 4.1: Formát dát IP geolokalizácie

Hneď po filtrovaní rádií je ďalším kľúčovým rysom podobných aplikácií vyhľadávanie, v tomto prípade vyhľadávanie názvov alebo častí názvov rádií. Tu sú na výber hneď dve možnosti, buď zahájiť vyhľadávanie až po zadaní hľadaného výrazu a explicitného stlačenia tlačidla hľadať alebo použiť inkrementálne vyhľadávanie, kde sa zoznam relevantných výsledkov dynamicky mení v závislosti na aktuálnom obsahu vyhľadávacieho poľa.
Vzhľadom k skutočnosti, že v aplikáciách pre telefóny Nokia je rozšírená predovšetkým druhá možnosť, voľba bola zrejmá. Avšak vyhľadávanie ako také so sebou prináša problém porovnávania výrazov, potenciálne zostavených zo znakov rôznych znakových sád. Konkrétne sa jedná o problém, keď užívateľ využíva iba základné znaky ASCII, ktoré mu v predvolenom nastavení ponúka virtuálna klávesnica telefónu, avšak napriek tomu sa snaží o vyhľadanie výsledkov, uchovávaných v databáze v ich plnej forme, teda obsahujú znaky s diakritikou, prípadne iné znaky špecifické pre rôzne národnosti.
Bez akejkoľvek transformácie vstupného výrazu nie je možné zaručiť očakávané výsledky. Je teda nejakým spôsobom nutné odstrániť z pôvodných názvov rádií diakritiku zo všetkých znakov. QML napriek svojej deklaratívnej paradigme povoľuje volanie funkcií jazyka JavaScript z akéhokoľvek miesta v zdrojovom kóde. Nanešťastie narozdiel od jazykov Java a C#, JavaScript neposkytuje žiadnu vstavanú funkciu na dekompozíciu znakov, a tým odstránenie diakritiky. Jazyky Java a C# poskytujú špecializované funkcie, ktoré sa operiajú o technickú správu konzorcia Unicode [4], ktorá definuje dva druhy ekvivalencie Unicode

[1] O zobrazovanie položiek do zoznamu sa v QML stará element *ListView*. Vizuálne zobrazenie jednotlivých položiek má za úlohu tzv. delegát (angl. *delegate*, ktorý definuje jednak parametre zobrazovanej položky, napr. výška a šírka oblasti, zarovnanie, oblasť reagujúcu na dotyk, ale aj akcie, ktoré sa vykonajú po kliknutí na konkrétnu položku.)

znakov. V kontexte riešeného problému je ale podstatný len prvý druh - kanonická ekvivalencia.
Táto ekvivalencia v stručnosti definuje ekvivalenciu znakov, prípadne sekvencie znakov, ak tieto reprezentujú identický abstraktný znak a zároveň sú identické pri korektnej vizuálnej reprezentácii. V rámci kanonickej ekvivalencie štandard definuje dve normalizačné formy, formu D pre dekompozíciu komplexného znaku na znak z ASCII a príslušnú diakritiku, a formu C pre kompozíciu sekvencie znakov na jeden komplexný znak vrátane diakritiky. Implementácia takejto normalizácie môže byť netriviálna, preto bola ako riešenie zvolená zámena problematických znakov za ich odpovedajúce ASCII ekvivalenty, anglicky sa táto operácie nazýva *string translation*. Ani na preklad však JavaScript nemá vstavanú funkciu, preto bola nevyhnutná vlastná implementácia. Prvým možným riešením je zostrojenie zoznamu slovíkov, kde by hodnotou prvého kľúča bol odpovedajúci regulárny výraz a hodnotou druhého kľúča príslušný ASCII ekvivalent. Potom by sa dala v slučke `for` volať funkcia replace s parametrami, regulárny výraz a odpovedajúci ASCII znak. Toto riešenie je ukázané na príklade 4.2

```
var translation = [
    {"regex":/[áâ...ä]/g,"letter":"a"},
    {"regex":/[ćç...č]/g,"letter":"c"},
    ⋮
    {"regex":/[ź...ž]/g,"letter":"c"},
];

function translate (inputStr) {
    for (var i=0; i < translation.length; i++) {
        outputStr = inputStr.replace(translation[i].regex,
           translation[i].letter);
    }
    return outputStr;
}
```

Příklad 4.2: Možné riešenie transformácie

Nakoniec však bol použitý variant inšpirovaný vstavanou funkciou *tr* z operačných systémov UNIX. Tá prijíma dva reťazce, prvý, ktorý definuje znaky nutné na preklad a druhý, ktorý definuje náhrady za znaky prvého reťazca. Fragment príslušného zdrojového kódu je ukázaný v príklade 4.3. Obe funkcie sú metódami elementu *SearchBar*, ktorý v okamihu zmeny obsahu vyhľadávacieho poľa volá metódu `compare(src1, src2)`, pričom jej predá porovnávané reťazce prevedené do notácie malých písmen, ktorá si porovnávané reťazce nechá najprv preložiť a potom ich porovná.

```
function translate(src) {
    var result = "";
    for (var i=0; i < src.length; i++) {  //najdi znak
        var pos = searchBox.trStr_in.search(src.charAt(i));
        if (pos !== -1) //ponechaj aktualny znak
            result += searchBox.trStr_out.charAt(pos);
        else result += src.charAt(i); //zamen znak
    }
```

```
    return result;  //vrat vysledok
}

function compare(src1,src2) {
    var str1 = translate(src1); //preloz hladany vyraz
    var str2 = translate(src2); //preloz nazov radia
    if (str2.search(str1) > -1) //je hladany vyraz podretazec?
        return true
    else return false
}
```

Příklad 4.3: Použitá transformácia

Podľa návratovej hodnoty metódy `compare(src1, src2)` sú potom relevantné rádiá vkladané do dedikovaného dynamického modelu, ktorého obsah sa musí pri každej zmene obsahu vyhľadávacieho panelu obnoviť.

4.2 Serverová časť - aplikácia typu SaaS

V nasledujúcej sekcii bude popísaný návrh a implementácia serverovej časti aplikácie, ako aj stručný popis zvolených implementačných prostriedkov, menovite webová vývojová nadstavba (ďalej len *framework*) Django, platforma OpenShift a databázový systém SQLite.

Použité nástroje

Ako prvý bude uvedený framework Django. Ako bolo spomenuté vyššie, jedná sa o vysokoúrovňový webový framework programovacieho jazyka Python, ktorý bol primárne vyvinutý pre pohodlný a výrazne rýchlejší vývoj jednoduchých až stredne pokročilých webových aplikácií. Uvedené črty možno pozorovať hneď v niekoľkých ohľadoch. Prvým je zaručene vstavaný administrátorský účet, ktorý je vygenerovaný ihneď po vytvorení projektovej sady súborov. Prístup k administrátorskému účtu je štandardne nastavený ako odpoveď na http požiadavok s URL sufixom */admin*, avšak toto nastavenie je možné zmeniť ako si ukážeme v neskorších odstavcoch. Po úspešnom prihlásení do administrátorského prostredia je možné spravovať ako jednotlivé databázové tabuľky (pre každú existuje individuálna položka v menu), tak aj užívateľské účty[2].
V rámci účtov sa jedná najčastejšie o operácie typu pridať, resp. odobrať určitú právomoc, prípadne užívateľský účet povýšiť až na administrátorský. Administrátorský účet taktiež prichádza s vlastnými grafickými šablónami a kaskádami, umiestnenými v hierarchii inštalačného adresára.
Ďalšou výhodou, ktorú Django prináša, sú databázové modely. Django odtieňuje nízkoúrovňový prístup k tabuľkám pomocou jazyka SQL syntaxou jazyka Python a definuje jednotlivé databázové modely ako triedy, dediace zväčša od základnej triedy *Model*. Od toho sa následne odvíja aj prístup k jednotlivým položkám tabuľky, pretože každý stĺpec výslednej tabuľky je reprezentovaný ako atribút príslušnej triedy. Ďalšie užitočné informácie primárne dokumentačného charakteru je možné dohľadať v [5].

Druhou spomedzi použitých nástrojov je platforma OpenShift. V kapitole 3 boli uvedené

[2]Podľa [5] sú heslá automaticky šifrované algoritmom PBKDF2 (angl. *Password-Based Key Derivation Function 2*) spolu s hašovacou funkciou SHA-256.

najpoužívanejšie modely služieb paradigmy Cloud computing. Platforma OpenShift sa radí do kategórie PaaS, čím jednak odtieňuje vývojára od infraštruktúry hardvéru nižšej vrstvy a ďalej mu vo forme jazykov, frameworkov a databázových systémov umožňuje sústrediť sa plne na vývoj aplikácie[3]. Jazyky a databázové systémy sú na platforme OpenShift dostupné v podobe abstraktných zásobníkov (angl. *cartridges*[4]). Systémové zdroje a zabezpečenie potom zaobstarajú špeciálne kontajnery zvané *gears*[5] a *nodes*[6] [12].
V prípade bezplatného umiestnenia aplikácie na platformu OpenShift dostane zákazník k dispozícii štandardne tri kontajnery (gears) typu *Small*[7], pričom ich môže použiť hneď troma rôznymi spôsobmi:

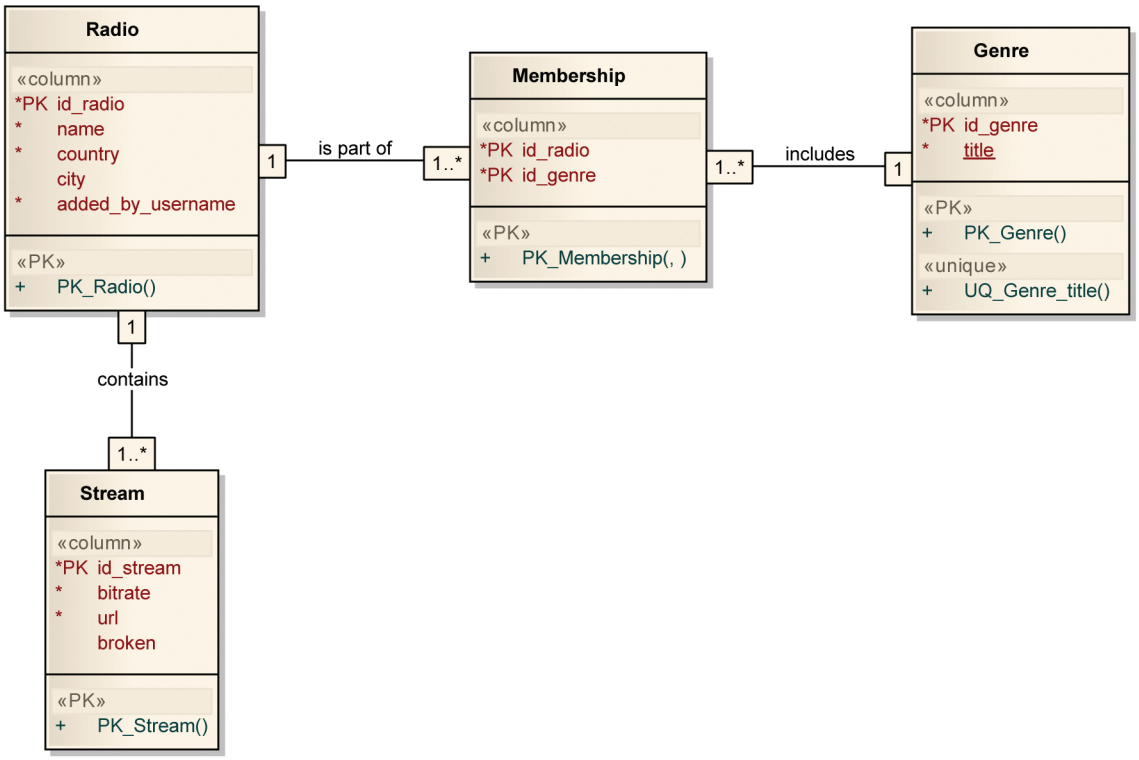
- využiť potenciál všetkých troch na jedinú aplikáciu a jej potrebné zásobníky
- využiť každý pre separátnu aplikáciu
- využiť kontajnery na škálovanie aplikácie (alokácia zdrojov na základe monitorovania záťaže)

Po vytvorení aplikácie v jej základnej podobe je pre ňu na serveri vytvorený repozitár. Pre samotný vývoj je však nutné si vytvoriť jeho lokálnu kópiu a jednotlivé odladené verzie aplikácie potom nahrávať na server (viď Nasadenie webovej aplikácie na OpenShift).
Posledným použitým nástrojom bol databázový systém SQLite, ktorý bol zvolený hlavne pre svoju jednoduchosť v používaní. V praxi sa v súvislosti s frameworkom Django odporúča použiť niektorý z pokročilejších databázových systémov (napr. MySQL), ale pre urýchlenie vývoja prvej verzie projektu je vhodné použiť práve databázový systém SQLite, ktorý možno kedykoľvek zmeniť v nastaveniach aplikácie.

Návrh

Ako hlavný požiadavok na serverovú časť aplikácie bolo stanovené budovanie a spolupráca s užívateľskou komunitou, ktorá by mala možnosť v prvom rade pridávať nové rádiá do existujúcej databázy a urýchliť tým rozširovanie repertoára streamov. Aby bolo možné sledovať aktivitu užívateľov, teda počty ich novo pridaných rádií, prípadne počty aktualizovaných rádií, ak tieto boli označené príznakom *Broken*, serverová časť aplikácie povinne vyžaduje registráciu.
Prvým bodom návrhu je vytvorenie modelu relačnej databázy. Ilustratívny model je uvedený na obrázku 4.3. Reprezentácia tabuliek už bola ukázaná v sekcii o použitých nástrojoch, teda tabuľky sú reprezentované ako triedy spolu s atribútmi. Zaujímavosťou je modelovanie vzťahu M:N. V takom prípade je zvolená jedna z tabuliek, ktorá bude nositeľom položky typu *ManyToManyField*, ktorá nám zaručí vzájomné prepojenie týchto tabuliek. V návrhu však bola z vizualizačných dôvodov administrátorského rozhrania využitá explicitná prepájacia tabuľka, ktorá je výhodná i v prípade, že model vyžaduje doplňujúce atribúty, ktoré nemožno jednoznačne priradiť ani jednej strane.
Nasledujúcim krokom je vizualizácia samotných tabuliek. V sekcii o použitých nástrojoch

[3] je nutné si uvedomiť, že PaaS poskytuje vývojové prostriedky, výsledná aplikácia určená pre koncového užívateľa je však typu SaaS
[4] Cartridges - nevyhnutné funkčné jednotky aplikácie.
[5] Gears - vyskytujú sa v podobe kontajnerov, ktoré automaticky obmedzujú využiteľné systémové zdroje dostupné jednému či viacerým zásobníkom.
[6] Nodes - ak je požadované zdieľanie zdrojov, niekoľko kontajnerov (gears) beží na jedinom fyzickom či virtuálnom stroji. Tieto stroje sú potom nazývané uzly (nodes).
[7] Small - zahŕňa využitie 512MB RAM, 100MB swap a 1GB diskového priestoru.



Obrázek 4.3: Databázový model

bol popísaný automatický administrátorský účet, vrátane prihlasovania kaskádových štýlov. Presmerovaním adresy so sufixom */admin* na koreňovú adresu sa vždy pri návšteve domovskej stránky zobrazí výzva na prihlásenie do internetovej databáze.

```
url(r'^', include(admin.site.urls))
```

Příklad 4.4: Django - presmerovanie administrátorskej stránky

Po prihlásení obyčajného užívateľa je mu zobrazené jednoduché menu spočívajúce v paneli posledných akcií, položky pre pridanie, resp. upravenie rádií a položky pre pridanie, resp. upravenie dostupných žánrov. Vizuálne elementy dostupné v náhľadoch na tabuľky sú definované v súbore *admin.py* formou administrátorských modelov, ktoré vizuálne reprezentujú databázové modely v rozhraní administrátora.

Aby sa užívatelia mohli do aplikácie prihlásiť, je potrebná ich registrácia. K tomu bol využitý voľne dostupný modul *django-registration* od Jamesa Bennetta[8]. Ten však počíta s dvojstupňovou registráciou, kedy sa pred aktiváciou užívateľského účtu odošle email s aktivačným odkazom, ktorý má určitú dobu expirácie. Tento krok bol z implementácie odstránený, teda účet je aktívny a prístupný ihneď po registrácii. Modul taktiež podporuje obnovu zabudnutého hesla na základe odoslaného emailu s unikátnym URL odkazom pre výber nového hesla. Pred ukončením registrácie sa novému užívateľskému účtu pridelia počiatočné privilégiá (táto črta bola doimplementovaná):

- pridávanie/zmena rádia
- pridávanie/zmena jednotlivých streamov konkrétneho rádia
- pridávanie/zmena žánrov

Modul bol taktiež prepojený s pôvodne administrátorskou možnosťou pre obnovu zabudnutého hesla. V takom prípade je na zvolenú adresu odoslaný email s unikátnym odkazom pre zmenu hesla a následné presmerovanie späť na koreňovú stránku.

Nasadenie webovej aplikácie na OpenShift

OpenShift podporuje verzovací systém Git, čo samotné nasadenie veľmi uľahčuje, pretože jediným príkazom `git push` sa všetky lokálne zmeny premietnu na server, automaticky dôjde ku štartu aplikácie, a tá je okamžite dostupná (napr. cez webový prehliadač). Problematickejšie však môžu byť prvotné správne nastavenia nasadzovania aplikácie na server. Práve z toho dôvodu bola do práce zaradená aktuálna sekcia.
Najdôležitejším súborom je aplikačný súbor *settings.py*. Ten obsahuje kompletnú konfiguráciu webovej aplikácie. Tieto konfigurácie zahŕňajú ladiaci režim, nastavenie databázového systému, internacionalizáciu, nastavenie adresárov so šablónami, nastavenie adresárov so statickými súbormi a správa modulov frameworku Django (tiež uvádzaných ako aplikácie), ktoré sú zodpovedné za fundamentálne operácie (menovite sprístupnenie automatického administrátora, sprístupnenie modulu umožňujúceho prácu so sedeniami, zozbieranie statických súborov).
Ladiaci režim je vhodný iba vo fáze testovania, pretože umožňuje náhľad na kompletné chybové hlásenie vrátane podrobného náhľadu na spätné trasovanie zásobníku volaní. Pri nasadení musí byť tento režim nutne vypnutý, v opačnom prípade bude aplikácia obsahovať zásadnú bezpečnostnú dieru. Ďalšie povinné nastavenie je nastavenie databázového systému. Django podporuje najrozšírenejšie databázové systémy: *MySQL, PostgreSQL, Oracle, SQLite*.
Pre potreby začiatočníkov alebo jednoducho pre potreby testovania sa odporúča ponechať nastavenie na systéme SQLite (vzorová aplikácia sa taktiež opiera o systém SQLite). Samozrejme v prípade potreby stačí vyplniť potrebné prihlasovacie údaje k databázovému serveru, príslušný databázový systém a Django zaručí bezproblémové prihlásenie. Šablóny podľa [5] definujú užívateľský vzhľad aplikácie. Django prináša špeciálny engine vrátane šablónovacieho jazyka, avšak šablóny možno vytvárať aj bez použitia syntaxe Python, iba so znalosťou jazyka HTML. *Settings.py* potom definuje, v akých adresároch bude server tieto šablóny vyhľadávať.
Posledným a najdôležitejším nastavením je nastavenie adresára statických súborov. Najprv

[8] https://bitbucket.org/ubernostrum/django-registration

k pojmu statický súbor. Webové stránky všeobecne potrebujú, aby hosťujúci server poskytoval obrázky, JavaScript moduly a CSS. Tieto sa súhrnne nazývajú statické súbory a je potrebný mechanizmus, ktorý by tieto súbory poskytoval vždy, keď je to nutné. Tento mechanizmus nazývame webserver. V základnom prevedení projektu vo frameworku Django sú všetky nastavenia príslušných statických adresárov prázdne.
Napriek tomu aplikácia na lokálnom testovacom serveri pobeží bez problémov, nakoľko Django je schopné tieto súbory dodať. Avšak podľa samotných vývojárov Django nikdy nebolo navrhnuté, aby priamo poskytovalo statické súbory a nahradilo úlohu servera (nakoľko server to dokáže oveľa rýchlejšie). K tomu slúži nastavenie `STATIC_ROOT`, ktoré definuje adresár (je nutné zadať absolútnu cestu), v ktorom sa bude uchovávať všetok statický obsah. V praxi sa odporúča tento adresár voliť na koreňovej úrovni, odlúčený od všetkých aplikačných adresárov.
To boli konfigurácie webovej aplikácie. Posledným krokom k úspešnému nasadeniu našej SaaS aplikácie je upravenie skriptu, ktorý sa spúšťa pri samotnom nasadzovaní na server. V adresári lokálneho repozitára[9] je potrebné pozmeniť skript *.openshift/action_hooks/deploy*. Obsah súboru je uvedený v prílohe A.

Aby sa zabránilo opätovnému kopírovaniu a inicializácii databázy, vzdialený repozitár má predurčený dátový adresár, ktorý bude obsahovať databázu a zabráni sa tak jej viacnásobnému nasadzovaniu. Pri prvom nasadení dôjde po vytvorení automaticky aj k inicializácii databázy. Nás však najviac zaujíma posledný príkaz `manage.py collectstatic`, kde dôjde k samotnému zozbieraniu všetkých potrebných statických súborov a ich umiestneniu do definovaného adresára (nastavenie `STATIC_ROOT`). Bez týchto zmien by aplikácia síce na serveri bežala, avšak v prípade akéhokoľvek požiadavku cez webový prehliadač by generovala internú chybu na serveri (chyba 500), pretože adresár so statickými súbormi by bol prázdny.
Na záver tejto sekcie o návrhu cloudovej časti aplikácie uvádzam odkaz do prílohy B, kde je uvedený obsah súboru *settings.py*, ktorý bol použitý pre nasadenie tejto aplikácie a môže tak slúžiť ako referenčná pomôcka.

4.3 Návrh aplikačného rozhrania, integrácia prehrávania do GUI

Pri návrhu API hral dôležitú úlohu výber sieťového protokolu. Taktiež bolo nutné vyriešiť formát posielaných dát (údajov o rádiách). Voľba padla na notáciu JSON, pretože jeho parsovanie je vo väčšine prípadov rýchlejšie ako druhá možnosť XML. JSON bol zvolený aj z dôvodu, že QML veľmi dobre spolupracuje s jazykom JavaScript, ktorý natívne používa JSON na dátovú reprezentáciu. Protokol by nemal byť zbytočne zložitý a mal by byť schopný prenášať serializované dáta v notácii JSON.
Nakoniec bol zvolený protokol HTTP, ktorého syntax veľmi dobre zapadá do celkového návrhu a Django pri návrhu webových aplikácii primárne uvažuje odpovede na HTTP požiadavky (najčastejšie vo forme formulárov). V prvom rade je nutné, aby aplikácia z mobilného zariadenia bola schopná získať požiadavkom dáta zo serveru. V tomto prípade aplikácia formuluje požiadavok typu `GET`, ako URL je predaná adresa internetovej databázy s cestou */radios/json*. Na strane serveru sú v prípade HTTP požiadavkov volané špeciálne funkcie súhrnne nazývané *views*. Funkcia `get_radio_json()` vracia objekt typu `HttpResponse`,

[9] lokálny repozitár sa vytvorí automaticky v aktuálnom adresári po vytvorení aplikácie cez príkazový riadok - `rhc app create` alebo v prípade vytvorenia cez webové rozhranie manuálne pomocou `git clone`

pričom do tela HTTP správy vloží zoserializovaný aktuálny zoznam aktívnych rádií. Práve serializácia je v prípade Djanga zaujímavá, pretože ten štandardne nepovoľuje serializáciu príbuzných tabuliek (spríbuznené pomocou cudzích kľúčov). Existuje však variant serializačného modulu v jazyku Python, zvaný `simplejson`. Tento variant umožňuje seralizovať vlastnoručne vytvorený reťazec (musí byť v korektnej notácii JSON). Uvádzam fragment kódu, ktorý realizuje vytvorenie vlastného reťazca v notácii JSON:

```
#do not serialize broken radios
status = self.is_broken()
if (status == 2):
    return ""
return {
    'id': self.id,
    'name': self.name,
    'country': self.country,
    'city': self.city,
    'genres': [{'id': genre.id, 'genre': genre.title} for
       genre in self.genre_set.all()],
    'streams': [{'id': stream.id, 'bitrate': stream.bitrate, '
       url': stream.url} for stream in self.stream_set.all()
       if not stream.broken],
}
```

Příklad 4.5: Django - serializácia databázy

Volanie metódy `is_broken` zaručuje, že rádiá, ktoré už nemajú žiaden funkčný stream a čakajú na aktualizáciu streamov sa budú serializovať ako prázdny reťazec, ktorý sa následne na strane aplikácie vynechá, a teda sa nezobrazí. Podobne je nutné vykonať kontrolu pri serializácii jednotlivých streamov, pretože niektoré streamy môžu byť pokazené, ale zvyšné streamy daného rádia sú plne funkčné, kontrola atribútu `broken` zaručí, že každé serializované rádio bude obsahovať iba aktívne streamy. Vo webovom rozhraní je navyše stav rádií odlíšený farebne[10], je teda ihneď zrejmé, ktoré rádiá je nutné aktualizovať a ktoré nie, tento fakt približuje obrázok 4.4.
Týmto sa dostávame k druhému typu požiadavku, `POST`, ktorý je využitý pri nahlásení nefunkčného streamu z rozrania klientskej aplikácie. Zvolený formát požiadavku je uvedený v nasledujúcom fragmente.

```
POST body:
    action_id : ID
    action_name : Action name
    id : radio_id
    url : broken stream url
```

Příklad 4.6: Formát požiadavku typu POST

Identifikátor a názov akcie sú dôležité z dôvodu, že v ďalších verziách projektu sa počíta so vzdialeným pridávaním, aktualizovaním a odstraňovaním rádií. Server je na túto situáciu pripravený deklaráciami príslušných funkcií. Tento krok však bude vyžadovať pridanie

[10] červená - rádio nemá žiaden aktívny stream a vyžaduje aktualizáciu
oranžová - rádio má aspoň 1 aktívny stream, zvyšok je neaktívnych
zelená - rádio má všetky streamy plne funkčné

registračného a prihlasovacieho modulu do samotnej klientskej aplikácie. Pre účely príkladovej aplikácie však táto funkcionalita nebola požadovaná a do návrhu nebola zahrnutá.

Posledným krokom bola integrácia prehrávacieho modulu do grafického rozhrania, popísaného v 4.1. Tento krok bol dosiahnutý komponentou *Audio* z multimediálnej knižnice *QtMultimediaKit*, ktorá sa najnižšej úrovni opiera o pluginy frameworku GStreamer, ktorý disponuje množstvom dekodérov pre rôzne formáty a v prípade streamovaného audio obsahu podporou RTP protokolu, ktorého teoretický základ bol položený v kapitole 2. V kontexte podporovaných formátov boli úspešne otestované formáty MP3, OGG a AAC.

Added by	Status
erzinho	Broken
mikez	Active
mikez	Active

Obrázek 4.4: Vizuálna reprezentácia stavu rádia v grafickom rozhraní webovej aplikácie

4.4 Zhodnotenie výslednej aplikácie

Výsledná klientská časť aplikácie bola po stránke užívateľského rozhrania a ovládania porovnávaná s existujúcim riešením aplikácie *Nobex Radio*, dostupnej na platformách Android a BlackBerry. Najväčšou nevýhodou Nobex Radio je komerčná licencia, pričom voľne dostupná verzia umožňuje prehrávať iba lokálne stanice. Vo svojom riešení som sa primárne zameral na opensource riešenie tohto problému a spoluprácu s komunitou, vďaka ktorej by bolo možné oveľa rýchlejšie rozširovanie databázy poskytovaných rádií.
V porovnaní s komerčnou verziou aplikácie Nobex prináša moja aplikácia pridávanie a odstraňovanie obľúbených staníc takmer z každého miesta aplikácie, čo by mohlo byť pre užívateľa rýchlejšie a pohodlnejšie riešenie ako pridávanie a odoberanie zo zoznamu obľúbených výhradne na hlavnej obrazovke. Prínosom by tiež mohla byť funkcia inkrementálneho vyhľadávania, ktorá vyhodnocuje aktuálne hľadaný reťazec a podľa neho zobrazuje relevantné výsledky hľadania (Nobex používa klasické vyhľadávanie reagujúce na potvrdenie).

Z pohľadu užívateľa pri testovaní aplikácie Nobex som nenarazil na možnosť nahlásenia nefunkčného streamu. Problém pri prehrávaní je oznámený iba dialógovým oknom, avšak tento stream naďalej zostáva v ponuke ako prehrávateľný. V prípade mnou implementovanej aplikácie je každý nefunkčný stream automaticky nahlásený bez vedomia užívateľa a pri ďalšom spustení aplikácie sa v ponuke už viac neobjavuje (ak nebol medzi tým opravený). Taktiež má užívateľ možnosť manuálneho nahlásenia či už nefunkčného streamu (zatiaľ nedetekovaného), primárne však streamu s potenciálne urážlivým charakterom.

Na aplikácii Nobex však treba veľmi pozitívne hodnotiť možnosť zdieľania informácií aktuálne počúvaného rádia na sociálnych sieťach, možnosť zaslania informácií o aktuálne hranom interpretovi či piesni na email a v neposlednom rade aj možnosť odkazu na videoklip k aktuálnej piesni na portále YouTube. Všetky tieto vlastnosti by mohli slúžiť ako inšpirácia do ďalšieho vývoja implementovanej aplikácie.
Do ďalšieho vývoja taktiež možno počítať s prenesením aplikácie na platformu BlackBerry, ktorá taktiež podporuje jazyk QML a jedinou zmenou by bola výmena špecifických kompo-

nent platformy MeeGo za komponenty platformy BlackBerry. Aplikácia by sa v budúcnosti mohla objaviť i v internetovom obchode Nokia Store.

Kapitola 5

Záver

Cieľom práce bolo vytvoriť aplikáciu, ktorá by demonštrovala problematiku streamovaného obsahu na mobilnom zariadení, pričom by sa opierala o dáta získané z cloudovej služby typu SaaS nasadenej na platformu OpenShift. Tento cieľ sa podarilo splniť.
Prvotným cieľom práce bolo preštudovanie problematiky prehrávania streamovanej hudby z bodu 1 zadania. Táto problematika bola diskutovaná v kapitole 2, predovšetkým sa však zaoberá protokolom RTP, jeho použitím a špecifikáciou. V súvislosti s RTP protokolom taktiež uvádza pojem payload formáty.
Druhým bodom zadania sa zaoberá kapitola 3, ktorá položila stručný teoretický základ do problematiky Cloud computingu, pričom primárne sa zameriava na službu typu SaaS, ktorej najvýraznejším rysom a zároveň výhodou je eliminácia poplatkov z pohľadu zákazníka za nákup softvérových licencií na lokálne počítače a samozrejme nákladov za beh a údržbu datacentier. Použitá literatúra z týchto bodov zadania je dohľadateľná v sekcii *Literatúra*.
Kapitola 4 je venovaná návrhu a implementácii užívateľského rozhrania, cloudovej služby typu SaaS a komunikačného rozhrania medzi klientskou a serverovou stranou. Pre streamovanie boli využité pluginy frameworku GStreamer ktoré sa opierajú i o protokol RTP spomínaný v kapitole 2. Pre webovú časť z bodu 3 zadania boli použité nástroje Django a platforma OpenShift. Zdrojové kódy k aplikácii sú dostupné na priloženom CD. Implementáciou aplikácie boli splnené body zadania 4 a 5.
Hlavným požiadavkom na klientskú časť aplikácie bolo, aby aplikácia využívala natívne komponenty danej platformy a sledovala jej zaužívané návrhové paradigmy. Aplikácia bola implementovaná pre platformu MeeGo a tento požiadavok bol splnený.

Špecifické vylepšenia do budúcnosti, ale i porovnania s existujúcim riešením a výsledky tejto práce boli diskutované v sekcii 4.4, vrátane možnosti prenesenia aplikácie na ďalšie platformy, predovšetkým BlackBerry. Hlavným prínosom aplikácie je určite možnosť pre užívateľov rozšíriť otvorenú internetovú databázu o obľúbené, avšak menej známe, rádiá. Prínosom je taktiež použitie stále sa rozvíjajúcej platformy OpenShift, na ktorej je hostovaná webová časť aplikácie, dostupná ako služba typu SaaS. Zhodnotením výsledkov aplikácie a ďalšieho pokračovania boli splnené všetky body zadania.

Literatura

[1] Bhardwaj, S.; Jain, L.; Jain, S.: Cloud Computing: A Study Of Infrastructure as a Service (IAAS). *International Journal of Engineering and Information Technology*, ročník 2, č. 1, 2010: s. 60–63, ISSN 0975-5292.

[2] Braden, R.; Clark, D.; Shenker, S.: Integrated Services in the Internet Architecture: an Overview. http://www.ietf.org/rfc/rfc1633.txt, 1994, [Online], [cit. 2013-05-03].

[3] Chen, W.; Lu, H.; Shen, L.; aj.: A Novel Hardware Assisted Full Virtualization Technique. In *2008 9th International Conference for Young Computer Scientists*, listopad 2008, ISBN 978-0-7695-3398-8, s. 1292–1297.

[4] Davis, M.; Whistler, K.: Unicode Normalization Forms. http://www.unicode.org/reports/tr15/, 2012, [Online], [cit. 2013-05-05].

[5] Django Software Foundation: Django documentation: Everything you need to know about Django. https://docs.djangoproject.com/en/1.4/, [Online], [cit. 2013-05-05].

[6] Foster, I. T.; Zhao, Y.; Raicu, I.; aj.: Cloud Computing and Grid Computing 360-Degree Compared. In *Grid Computing Environments Workshop. 2008. GCE '08*, listopad 2008, ISBN 978-1-4244-2860-1, s. 1–10.

[7] Kurose, J. F.; Ross, K. W.: *Computer Networking: A Top-Down Approach Featuring the Internet*. Addison-Wesley, třetí vydání, 2005, ISBN 0-321-22735-2.

[8] Lawton, G.: Developing Software Online with Platform-as-a-Service Technology. *Computer*, ročník 41, č. 6, červen 2008: s. 13–15, ISSN 0018-9162.

[9] Matyska, L.: Techniky virtualizace počítačů (2). *Zpravodaj ÚVT MU*, ročník 17, č. 3, 2007: s. 9–12, ISSN 1212-0901.

[10] Nichols, K.; Blake, S.; Baker, F.; aj.: Definition of the Differentiated Services Field (DS Field) in the IPv4 and IPv6 Headers. http://www.ietf.org/rfc/rfc2474.txt, 1998, [Online], [cit. 2013-05-03].

[11] Perkins, C.: *RTP: audio and video for the internet*. Boston: Addison-Wesley, 2003, ISBN 0-672-32249-8.

[12] Red Hat: OpenShift All Versions User Guide. https://www.openshift.com/sites/default/files/documents/OpenShift-2.0-User_Guide-en-US_5.pdf, [Online], [cit. 2013-05-05].

[13] Sabahi, F.: Virtualization-level security in cloud computing. In *2011 IEEE 3rd International Conference on Communication Software and Networks*, květen 2011, ISBN 978-1-61284-485-5, s. 250–254.

[14] Schulzrinne, H.; Casner, S.: RTP: A Transport Protocol for Real-Time Applications. http://www.ietf.org/rfc/rfc3550.txt, 2003, [Online], [cit. 2013-02-20].

[15] Sun, W.; Zhang, X.; Guo, C. J.; aj.: Software as a Service: Configuration and Customization Perspectives. In *2008 IEEE Congress on Services Part II*, září 2008, ISBN 978-0-7695-3313-1, s. 18–25.

[16] Tanenbaum, A.; Wetherall, D.: *Computer Networks*. Prentice Hall, páté vydání, 2010, ISBN 978-0-13-212695-3.

[17] Velte, A.; Velte, T. J.; Elsenpeter, R. C.: *Cloud Computing: A Practical Approach*. McGraw Hill Proffesional, 2009, ISBN 978-0-07-162695-8.

[18] VMware, Inc.: VMware White Paper: Understanding Full Virtualization, Paravirtualization, and Hardware Assist. http://www.vmware.com/files/pdf/VMware_paravirtualization.pdf, [Online], [cit. 2013-03-24].

Príloha A

Konfiguračný súbor deploy

```
#!/bin/bash
cartridge_type="python-2.6"
source $OPENSHIFT_HOMEDIR/$cartridge_type/virtenv/bin/activate

if [ ! -f $OPENSHIFT_DATA_DIR/sqlite.db ]
then
    echo "Copying $OPENSHIFT_REPO_DIR/wsgi/radiostream/sqlite.
       db to $OPENSHIFT_DATA_DIR"
    cp "$OPENSHIFT_REPO_DIR"wsgi/radiostream/sqlite.db
       $OPENSHIFT_DATA_DIR
    python "$OPENSHIFT_REPO_DIR".openshift/action_hooks/
       secure_db.py | tee ${OPENSHIFT_DATA_DIR}/CREDENTIALS
else
    echo "Executing 'python $OPENSHIFT_REPO_DIR/wsgi/
       radiostream/manage.py syncdb --noinput'"
    python "$OPENSHIFT_REPO_DIR"wsgi/radiostream/manage.py
       syncdb --noinput
fi

echo "Executing 'python $OPENSHIFT_REPO_DIR/wsgi/radiostream/
   manage.py collectstatic --noinput'"
python "$OPENSHIFT_REPO_DIR"wsgi/radiostream/manage.py
   collectstatic --noinput
```

Príloha B

Konfiguračný súbor settings.py

```
import os

ON_OPENSHIFT = False
if os.environ.has_key('OPENSHIFT_REPO_DIR'):
    ON_OPENSHIFT = True
PROJECT_DIR = os.path.dirname(os.path.realpath(__file__))

if ON_OPENSHIFT:
    DEBUG = False
else:
    DEBUG = True

TEMPLATE_DEBUG = DEBUG

ADMINS = (
    # ('Your Name', 'your_email@example.com'),
)
MANAGERS = ADMINS

if ON_OPENSHIFT:
    DATABASES = {
        'default': {
            'ENGINE': 'django.db.backends.sqlite3',  # Add '
               postgresql_psycopg2', 'postgresql', 'mysql', '
               sqlite3' or 'oracle'.
            'NAME': os.path.join(os.environ['
               OPENSHIFT_DATA_DIR'], 'sqlite.db'),  # Or path
               to database file if using sqlite3.
            'USER': '',                      # Not used with
               sqlite3.
            'PASSWORD': '',                  # Not used with
               sqlite3.
            'HOST': '',                      # Set to empty
               string for localhost. Not used with sqlite3.
```

```
            'PORT': '',                      # Set to empty
               string for default. Not used with sqlite3.
        }
    }
else:
    DATABASES = {
        'default': {
            'ENGINE': 'django.db.backends.sqlite3',  # Add '
               postgresql_psycopg2', 'postgresql', 'mysql', '
               sqlite3' or 'oracle'.
            'NAME': os.path.join(PROJECT_DIR, 'sqlite.db'),  #
                Or path to database file if using sqlite3.
            'USER': '',                      # Not used with
               sqlite3.
            'PASSWORD': '',                  # Not used with
               sqlite3.
            'HOST': '',                      # Set to empty
               string for localhost. Not used with sqlite3.
            'PORT': '',                      # Set to empty
               string for default. Not used with sqlite3.
        }
    }

TIME_ZONE = 'Europe/Bratislava'
LANGUAGE_CODE = 'en-us'
SITE_ID = 1
USE_I18N = True
USE_L10N = True
USE_TZ = True
MEDIA_ROOT = os.environ.get('OPENSHIFT_DATA_DIR', '')

# URL that handles the media served from MEDIA_ROOT.
MEDIA_URL = ''
STATIC_ROOT = os.path.join(PROJECT_DIR, '..', 'static')
STATIC_URL = '/static/'
STATICFILES_DIRS = (
)

STATICFILES_FINDERS = (
    'django.contrib.staticfiles.finders.FileSystemFinder',
    'django.contrib.staticfiles.finders.AppDirectoriesFinder',
)

SECRET_KEY = '1j7+_i*)y3+v^k1015rzmawpsotga$_^im3y7wxc5m0r@)l_
   &amp;('

TEMPLATE_LOADERS = (
    'django.template.loaders.filesystem.Loader',
```

```
    'django.template.loaders.app_directories.Loader',
)

MIDDLEWARE_CLASSES = (
    'django.middleware.common.CommonMiddleware',
    'django.contrib.sessions.middleware.SessionMiddleware',
    'django.middleware.csrf.CsrfViewMiddleware',
    'django.contrib.auth.middleware.AuthenticationMiddleware',
    'django.contrib.messages.middleware.MessageMiddleware',
)

ROOT_URLCONF = 'radiostream.urls'
WSGI_APPLICATION = 'radiostream.wsgi.application'

TEMPLATE_DIRS = (
    os.path.join(PROJECT_DIR, 'templates'),
)

INSTALLED_APPS = (
    'django.contrib.auth',
    'django.contrib.contenttypes',
    'django.contrib.sessions',
    'django.contrib.sites',
    'django.contrib.messages',
    'django.contrib.staticfiles',
    'django.contrib.admin',
    'radios',
    'registration',
)
LOGGING = {
    'version': 1,
    'disable_existing_loggers': False,
    'handlers': {
        'mail_admins': {
            'level': 'ERROR',
            'class': 'django.utils.log.AdminEmailHandler'
        }
    },
    'loggers': {
        'django.request': {
            'handlers': ['mail_admins'],
            'level': 'ERROR',
            'propagate': True,
        },
    }
}
```